PROJ2_DataDrillers

by MULYANI BINTI SARIPUDDIN A22EC0223

Submission date: 06-Jul-2025 11:05PM (UTC-0700)
Submission ID: 2711282180
File name: PROJ2_DataDrillers.pdf (2.23M)
Word count: 8690
Character count: 52721

UTM
UNIVERSITI TEKNOLOGI MALAYSIA
Department of Computer Science
Faculty of Computing

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

Group B - Data Drillers

Subject Code	:	SECP3133
Subject Name	:	HIGH PERFORMANCE DATA PROCESSING
Session-Sem	:	24/25-2

Prepared by : 1) MUHAMMAD ANAS BIN MOHD PIKRI (A21SC0464)
2) MULYANI BINTI SARIPUDDIN (A22EC0223)
3) ALIATUL IZZAH BINTI JASMAN (A22EC0136)
4) THEVAN RAJU A/L JEGANATH (A22EC0286)

Section : 01

Lecturer : Dr Mohd Shahizan Bin Othman

Date : 7 May 2025

Table of Contents

1.0 Introduction

In the current dynamic world of the digital age, it has become extremely important to know what people say in real time so that organisations and companies can make effective decisions to counter any market forces coming to play. The interesting subject matter of this project is real-time sentiment analysis, where a pipeline to extract the data, process, and visualise the thoughts of the masses in the digital media becomes critical.

Our main task will be to create and code a system of real-time sentiment analysis through the capabilities of Apache Spark in processing big data and Apache Kafka in the ability to stream the data with no interruptions. In particular, we are going to conduct the analysis of user sentiments in comments left in the review section of the Google Play Store on such Malaysian e-wallet and e-commerce applications as Touch n Go, Boost, Grab, Setel, and Shopee. It will be a focused strategy that will educate us about the kind of perceptions that the user holds towards such commonly used services in the Malaysian setting.

The pipeline is expected to include the following important steps: raw review data are streamed into Kafka and the advanced Natural Language Processing (NLP) methods are used to clean data, tokenise it, and generate features; then, the machine learning or deep learning models can be deployed to perform sentiment classification (positive, negative, or neutral). Lastly, the progressive real-time details will be delivered in an interactive, user-friendly dashboard developed with Streamlit to allow stakeholders to intuitively understand the direction of sentiments and actionable insights. Not only will such a project focus on delivering a functional real-time analytics solution it will also involve a practical experience in learning key skills in big data engineering and processing as well as data visualization.

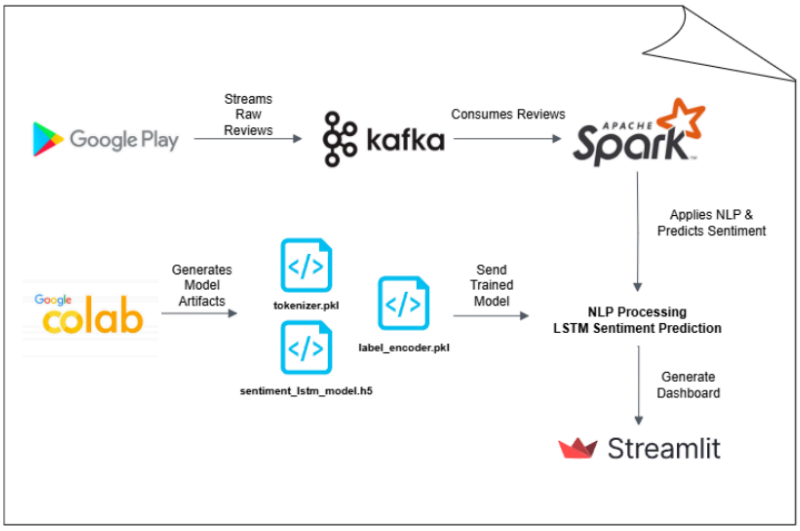


Figure 1: Overview Diagram of Workflows

2.0 Data Acquisition & Preprocessing

In this project, the primary source of data is the Google Play Store, where users publicly post reviews and ratings for various mobile applications. To collect this data in real-time, we used a custom Python script powered by the Google-Play-Scraper library, which is designed to extract app metadata and user reviews by querying the Google Play API.

2.1 Target Application

The applications selected for sentiment analysis were popular Malaysian e-wallet and e-commerce apps:

App Name	Package Name
Touch'N Go	"my.com.tngdigital.ewallet"
Boost	"my.com.myboost"
GrabPay	"com.grabtaxi.passenger"
Setel	"com.setel.mobile"
Shopee Pay	"com.shopee.pay"

Data was collected using the Python library Google-Play-Scraper, which supports multilingual review extraction and metadata capture. Only Bahasa Melayu (lang='ms') reviews were extracted to ensure local sentiment relevance.

2.2 Scraping Methodology

Link for Google scraping: preprocessing.ipynb

Methods	Description
Library	google-play-scraper
Platform	Python (Google Colab runtime)
Batch Size	200 reviews per request

Sort Order	Newest first
Automation	Integrated with a Kafka producer for real-time or batched ingestion

Each extracted review was structured into a dictionary with the following fields:

1. `app` : App name
2. `username` : Name of the reviewer
3. `review` : Raw text of the review
4. `rating` : Star rating (1-5)
5. `date` : Date with the review was posted

Sentiment Labeling Logic (Rule-based) :

Rating	Sentiment
4-5	Positive
3	Neutral
1-2	Negative

2.3 Initial Preprocessing

Before applying NLP techniques and feeding the data into Apache Spark, the reviews underwent an initial preprocessing stage:

Steps Involved :

1. **Lowercasing :** Standardises all text to lowercase
2. **Emoji Removal :** Used emoji.replace_emoji() to eliminate emojis.
3. **Punctuation & Digit Removal :** Regex was applied to retain only alphabetic characters.
4. **Whitespace Normalization :** Removed extra spaces and line breaks
5. **Stopword Filtering :** English stopwords were removed using NLTK
6. **Date Parsing :** Converted spring dates to datetime format, handling inconsistencies.
7. **Empty Review Filtering :** Dropped any null or whitespace-only reviews.

3.0 Sentiment Model Development

3.1 Overview

Two models were trained and evaluated as part of executing the real-time sentiment classification.

Purpose	Model	Strengths	Streaming Use-Case
Fast & lightweight	TF-IDF + Multinomial Naive Bayes	High speed, low latency, Spark-native friendly	Full real-time classification inside Spark
Deep context-aware	LSTM Neural Network (Keras)	Captures the word sequence and nuance of the Malay Language	Used in the current Spark Kafka pipeline

The dual-model can be compared in terms of such trade-offs as speed and accuracy, and is also flexible in as much as it depends on the system requirements.

3.2 Dataset & Labeling

Item	Detail
Source	cleaned_reviews.csv scraped via google-play-scraper
Apps	Touch ‘n Go, Boost, GrabPay, Setel, Shopee
Language Filter	Bahasa Melayu (lang='ms')
Review Fields Used	review (text), rating (for sentiment)
Label Logic (Rule-based)	Star rating 4–5 → Positive, 3 → Neutral, 1–2 → Negative
Preprocessing	Lowercase, emoji & punctuation removal, stopword filtering, whitespace normalization (Done in previous step)
Tooling	Colab with NLTK, regex, emoji, sklearn, TensorFlow/Keras

Through this rule-based labeling, it was achievable to train a huge data set without the help of manual labeling.

3.3 Model 1 – Naive Bayes (TF-IDF)

```
vectorizer = TfidfVectorizer(max_features=5000)
model = MultinomialNB()
```

Setting	Value
Features	Top 5,000 words (TF-IDF)
Train/Test Split	80/20
Accuracy	**87.73%**
Macro F1 Score	**0.55**
Runtime	9 seconds

The Naive Bayes model uses a TF-IDF vector of the review texts and performs well on positive and negative sentiments. It performs poorly at neutral class as it is underrepresented (~4 percent of data).

Classification Report Highlights:

- Negative F1: 0.71
- Neutral F1: 0.00 (very low because of imbalance)
- Positive F1: 0.93

This model is used for **ultra-low latency** inference. It can be broadcast into Spark executors for batch micro-processing or used in parallel with the LSTM model.

3.4 Model 2 – LSTM (Keras)

```
model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),
    LSTM(64),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])
```

Setting	Value
Tokenizer Vocabulary	10,000 words
Sequence Length	100 tokens
Accuracy	**88.00%**
Macro F1 Score	**0.57**
Runtime	11 minute
Artifacts Saved	sentiment_lstm_model.h5, tokenizer.pkl, label_encoder.pkl

The training of this model was done on the same preprocessed data on reviews. While it also suffers on the neutral class, the LSTM has slightly better performance than Naive Bayes.

Classification Report Highlights:

- Negative F1: 0.71
- Neutral F1: 0.05
- Positive F1: 0.94

3.5 Training and Optimization Workflow

Step	Description
Train/Test Split	80/20 split
Tokenization	Used Keras Tokenizer (num_words=10000)
Padding	Fixed sequence length = 100
Regularization	Dropout (0.5)
Model Evaluation	Used accuracy & macro-averaged F1

These two models were trained through Google Colab. The LSTM model with its tokenizer as well as label encoder were saved and used in Spark streaming for production.

3.6 Integration into Spark Streaming

The **LSTM model** is currently deployed directly inside the Spark Kafka consumer script:

```
@udf(StringType())
def predict_sentiment(text):
    if not text:
        return "neutral"
    sequence = tokenizer.texts_to_sequences([text])
    padded = pad_sequences(sequence, maxlen=100)
    prediction = model.predict(padded)
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

df_with_sentiment = df_parsed.withColumn("predicted_sentiment",
predict_sentiment(col("review")))
```

Execution flow:

1. Kafka streams in raw JSON reviews.
2. Spark parses and retrieves the field review.
3. The LSTM model is applied on every row in the memory of UDF.
4. It is followed by predicted sentiment that is attached to each row of review.
5. Results are already outputted to a console but can be forwarded to Streamlit.

3.7 Model Comparison Summary

Metric	Naive Bayes	LSTM (Keras)
Accuracy	87.73%	88.00%
Macro F1	0.55	0.56
Neutral F1	0.00	0.04
Latency (avg)	~0.09 ms	~6.55 ms
Deployment Method	Broadcast UDF or REST	UDF (in-memory)

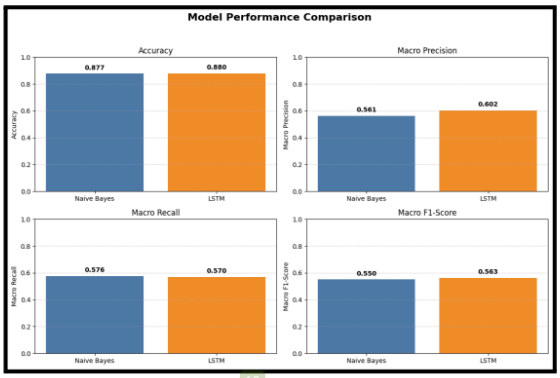


Figure 3: *Model Performance Comparison based on Accuracy, Precision, Recall, and F1-Score. LSTM shows slightly better results across most evaluation metrics, while Naive Bayes performs slightly better in macro recall.*

Conclusion:

LSTM offers slightly higher accuracy and better understanding of Malay syntax and slang, but Naive Bayes is much faster and easier to scale for high-throughput pipelines. Although Naive Bayes was impressive in terms of speed of inference and overall accuracy, we decided to implement the LSTM model in our Spark Kafka pipeline because it would deal with the Bahasa Melayu structure and context more effectively than Naive Bayes, and slightly better on all metrics, particularly the Neutral class. Both models were kept as evaluation models, and LSTM was chosen in the final implementation of the system as running in real-time sentiment analysis.

3.8 Limitations & Future Work

- **Neutral class performance** remains low as the labeled data is limited so the future work could involve:
    - Manual annotation
    - Synthetic sample generation
- **Malay slang handling**: Performance can improve with contextual embeddings (e.g., Malay BERT)
- **Scalability**: Consider deploying LSTM as a REST microservice to reduce Spark memory load

3.9 Model Artifacts

File	Description
sentiment_nb.pkl	Pickled TF-IDF + NB model
sentiment_lstm_model.h5	Trained LSTM model
tokenizer.pkl	Word token-to-ID mapping
label_encoder.pkl	Sentiment label encoder

All models will be saved in Google Drive → Project2 folder and versioned so that they can be reusable.

3.10 Colab Notebook Reference

This process of both model development and training was carried out in Google Colab with Python.The notebook contains:

- Data loading & preprocessing
- Naive Bayes + TF-IDF model training
- LSTM model training with TensorFlow
- Evaluation results and saved artifacts

View Colab Notebook

4.0 Apache System Architecture

4.1 Apache Kafka

Role: Apache Kafka plays the role of serving as the key foundation in real-time data ingestions, as a high-throughput, fault-tolerant, and streaming platform. It functions as the central nervous system in controlling the data streams from a source to processing.

Key Components and Functions:

- Producers: The Google Play Store scraper is a Kafka Producer. Its role is to fetch raw review feedback of the e-wallets and post each feedback as an individual message to a given Kafka topic.

Advantages:

- Decoupling: Kafka does a good job at decoupling consumers and data producers and enables them to work independently and scale separately.
- Backpressure Handling: It is extremely efficient in handling back pressure, thus keeping the systems free of pressure in case there is excessive data.
- Data Durability: Messages are also stored on a disk such that data will not get misplaced when consumers or producers fail.
- Scalability: Is capable of handling many consumers reading on a single data stream at once, with the overhead not interfering with each other's progress.

4.1.1 Setting Up and Running Apache Kafka

Requirements:

- Java Development Kit (JDK): Kafka requires Java. Download and install a compatible JDK
  https://openjdk.org/install/
- Apache Kafka: Download the latest stable release of Apache Kafka
  https://kafka.apache.org/downloads
  (Choose a binary download.)

Setup Steps:

- Extract Kafka: Unzip the downloaded Kafka archive to a directory (C:\kafka on Windows or /opt/kafka on Linux/macOS).
- Navigate to Kafka Directory: Open your command prompt or terminal and navigate to the extracted Kafka directory.

```
# On Windows
cd C:\kafka

# Or on Linux/macOS
cd /opt/kafka
```

1. **Start Kafka Zookeeper**
   Open a command prompt and write the command below:

```
# On Windows
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

# On Linux/macOS bin/zookeeper-server-start.sh config/zookeeper.properties
```

2. **Start Kafka Broker**

   Without closing the previous one, open another new command prompt and write the command below:

```
# On Windows
.\bin\windows\kafka-server-start.bat .\config\server.properties

# On Linux/macOS
bin/kafka-server-start.sh config/server.properties
```

3. **Create a Kafka Topic**

   Without closing the previous one, open another new command prompt and write the command below:

```
# On Windows
.\bin\windows\kafka-topics.bat --create --topic raw_reviews_topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

# On Linux/macOS
.\bin/kafka-topics.sh --create --topic raw_reviews_topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
```

4.1.2 Source Code

Kafka Live Producer (kafka_live_producer.py)

A constant scraping script of the reviews found on the Google Play Store related to the given e-wallet applications existing on the platform, and reporting it in JSON messages to a topic.

Kafka Setup

- Purpose: The section initializes the Kafka producer.
- KafkaProducer: A KafkaProducer instance is instantiated and it works to transmit the messages to Kafka brokers.
- bootstrap_servers='localhost:9092' means the Kafka broker(s) that the producer will be able to connect to. Here it will be configured to a local Kafka instance that is started on port 9092.

```
from kafka import KafkaProducer
# ... other imports ...

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

Apps to Track

- Purpose: This dictionary contains names and package IDs of the e-wallet apps whose reviews are going to be scraped.
- Structure: Keys are all friendly apps names (e.g., 'TouchNGo'), and values are the package names that Google Play Store uses to identify their unique content (e.g., 'my.com.tngdigital.ewallet').

```
#       Apps to track
apps = {
    'TouchNGo': 'my.com.tngdigital.ewallet',
    'Boost': 'my.com.myboost',
    'Grab': 'com.grabtaxi.passenger',
    'Setel': 'com.setel.mobile',
    'Shopee': 'com.shopeepay.my', }
```

Sent Review Cache to Avoid Duplicates

- sent_reviews = set (): This makes a blank Python set. The set is applied due to its mechanism to store the individual elements which is also effective when determining whether a review had been reviewed already and passed to Kafka. This comes in handy to avoid duplicate reviews that may occur when the scraper retrieves the same review severally in different runs or even during a run.
- make_review_id(app, username, date): This is a helper function that creates a unique id of each review by combining app name, username and the date of the review. This ID is in turn tracked in the sent_reviews set to track processed reviews.

```
#       Sent review cache to avoid duplicates
sent_reviews = set()

#       Helper: Generate unique ID
def make_review_id(app, username, date):
   return f"{app}_{username}_{date}"
```

Main Real-time Streaming Loop

- r.get('content'): continue: Exits those reviews that were found to contain no content.
- review_date = r['at'].strftime('%Y-%m-%d %H:%M:%S'): Formats the review timestamp into a readable string.
- review id = make review id(...): Brings about the creation of an ID of the current review.
- if review_id in sent_reviews: continue: Checks whether the given review has been sent by a user. In that case, it moves to the next review so as to prevent duplicates.
- review_data: {...}: They create a dictionary with pertinent data about the review: name of the app, user name, content of the review (leading/trailing whitespace removed), rating (score), date of the review and timestamp of when the review was scraped (fetched_at).
- producer. --sendIt_('review_topic', value=review_data): This is at the centre of Kafka action. The dictionary review_data (which will be converted to JSON by the value_serializer will be sent to the Kafka topic with the name review_topic).
- sent_reviews[review_id]: The review_id of the sent review is put in.
- print(...): Printed to the console will be a confirmation message of each review.
- time.sleep(60): Once the program has completed review processing of all apps, it waits 60 seconds then begins another fetch cycle in order to avoid dropping its connection to

the Google Play Store server and flooding the servers because it will cause users to not see any reviews and/or app page may take too long to load and instead achieve real time streaming of new reviews almost instantaneously with few requests being made relative to making up to thousands.

```
print("      Starting real-time streaming to Kafka (every 60s)...\n")
while True:
    for app_name, package_name in apps.items():
        try:
            result, _ = reviews(
                package_name,
                lang='ms',   # Bahasa Melayu only
                country='my',
                sort=Sort.NEWEST,
                count=100  # max possible to catch up on latest reviews
            )

            for r in result:
                if not r.get('content'):
                    continue

                review_date = r['at'].strftime('%Y-%m-%d %H:%M:%S')
                review_id = make_review_id(app_name, r['userName'], review_date)

                if review_id in sent_reviews:
                    continue

                review_data = {
                    'app': app_name,
                    'username': r['userName'],
                    'review': r['content'].strip(),
                    'rating': r['score'],
                    'date': review_date,
                    'fetched_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                }

                producer.send('review_topic', value=review_data)
                sent_reviews.add(review_id)

                print(f"      Sent: [{app_name}] {r['userName']} ({r['score']}) - {review_date}")

        except Exception as e:
            print(f"      Error scraping {app_name}: {e}")

    print("      Sleeping 60 seconds before next fetch...\n")
```

```
time.sleep(60)
```

4.2 Apache Spark Streaming

Role: Apache Spark Streaming adds the core Spark API with a set of extensions to allow scalable, high-throughput and fault-tolerant processing of live data streams. It consumes raw data, does analysis on it, some of which is complicated and generates data ready to be stored and visualised.

Key Components and Functions:

- Consumers: A special application written using Spark Streaming is a consumer of Kafka. It subscribes to raw_reviews_topic and constantly pulls new messages containing review comments to process them out of Kafka.
- Processing Model: Spark Streaming works with the data in tiny, time-centred gatherings (micro-batches). There are more recent versions of Spark, which provide continuous processing to meet lower latency needs.

Processing Steps within Spark:

- The pre-trained LSTM model (sentiment_lstm_model.h5), its related tokens.pkl (tokenizer.pkl) and its tag encoder.pkl (label_encoder.pkl) are imported to the Spark app.
- A User-Defined Function (UDF) is defined, predict_sentiment which is then applied on review column of the arriving data.
- Within this UDF, raw review text is put through preprocessing steps needed as an input to LSTM model:
- Tokenization: The loaded tokenizer is used to convert the text in to a series of number.
- Padding: Such sequences are activated to a standard length (maxlen=100) so as to comply with requirements such as LSTM entries.
- The loaded LSTM model is then called, inputted with the padded sequences in order to give the sentiment predictions.
- Instead the numerical output is returned to the human readable sentiment labels (e.g. positive, negative, neutral) through the label_encoder.

- A new column (predicted_sentiment) with the predicted sentiment is introduced in every single review record.

Advantages:

- Scalable: Handles bulk data over the clusters with efficacy.
- Fault Tolerance: Can minimise the loss of data through failures.
- Real-time Processing: Processes live streams in real-time and fast.
- Unified Analytics Engine: Supports various workloads such are batch, streaming, Machine Learning (ML) and Graph.

4.2.1 Setting Up and Running Apache Spark

Requirements:

- Java Development Kit (JDK): Spark requires Java 8 or newer.
  https://openjdk.org/install/
- Apache Spark: Download a pre-built package of Apache Spark with Hadoop.
  https://spark.apache.org/downloads.html
  (Choose a pre-built package for Hadoop 3.3 or similar.)
- Python: Ensure you have Python 3.7+ installed.
  https://www.python.org/downloads/

Setup Steps:

- Extract Spark: Unzip the downloaded Spark archive to a directory (e.g., C:\spark on Windows or /opt/spark on Linux/macOS).
- **Set Environment Variables:**
  - **SPARK_HOME:** Set this to the path where you have extracted the Spark file (e.g., `C:\spark\spark-3.5.1-bin-hadoop3`) .
  - **PATH:** Add `%SPARK_HOME%\bin` (Windows) or `$SPARK_HOME/bin` (Linux/macOS) to your system's PATH variable.
  - **HADOOP_HOME**: If on Windows, you are required to set `HADOOP_HOME` and place `winutils.exe` in `%HADOOP_HOME%\bin`.
  - **JAVA_HOME:** Ensure `JAVA_HOME` is set to your JDK installation path.
- Run Streaming Application via PySpark

```
/path/to/file/sentiment_streaming_app.py
```

4.2.2 Source Code

Spark Kafka Consumer (spark_kafka_consumer.py)

The script will pull review data off of Kafka and then process it using a pre-trained LSTM to perform sentiment prediction; it will then print the output into the console. In Thi the author uses Apache Spark Structured Streaming to ingest real-time data about reviews in a Kafka topic, analyse the reviews on their sentiment with a pre-trained Keras LSTM model, and ultimately emit its analysis. Such implementation is common in real-time data processing pipelines where data is received in a continuous way.

1. Imports and Model/Tokenizer Loading

- Purpose: In this section, imports are made, and the pre-trained machine learning components needed in performing the sentiment analysis are loaded.
- pyspark.sql imports: These are critical in accessing Spark DataFrames as well as the creation of User Defined functions (UDFs).
- tensorflow: Was applied in loading and running the pre-trained model of deep learning.
- pickle: It was used to load (deserialise) the tokeniser and label_encoder object,s which were saved.
- numpy: It is to be used in numerical computations, namely, np.argmax to retrieve the index of maximum probability in the prediction of the model.
- tokenizer.pkl: This is a file and has a tokenizer of type tf.keras.preprocessing.text. This tokeniser plays a critical role since it translates raw text reviews to numerical sequences, which could be comprehended by the LSTM model. It translates words to integer numbers using the vocabulary acquired during the training of the model.
- label_encoder.pkl: Here is a Python pickled file which contains sklearn. preprocessing.LabelEncoder object. It inverts the numerical labels of sentiments (e.g. 0, 1, 2) to their original string form (e.g. positive, neutral, negative).
- sentiment_lstm_model.h5: It is the pre-trained Long Short-Term Memory (LSTM) neural network model with the file format of HDF5. The given model is in charge of manipulating the program to extract the numerical chains of review text and predicts the sentiment of those texts.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, udf
from pyspark.sql.types import StructType, StringType
import tensorflow as tf
import pickle
import numpy as np

# Load tokenizer and encoder from local path
with open('tokenizer.pkl', 'rb') as f:
    tokenizer = pickle.load(f)

with open('label_encoder.pkl', 'rb') as f:
    label_encoder = pickle.load(f)

# Load trained LSTM model
model = tf.keras.models.load_model('sentiment_lstm_model.h5')
```

2. Spark Session Creation

- Purpose: Creates and returns a SparkSession object that contains basic functionality of using Spark.
- SparkSession.builder: Makes it possible to configure and create a SparkSession.
- appName("KafkaReviewConsumer"): name of the Spark application assists in identifying the application in the Sparks UI.
- getOrCreate(): Gets an existing SparkSession and creates a new one in case it does not exist.

```
# Create Spark session with Kafka support
spark = SparkSession.builder \
    .appName("KafkaReviewConsumer") \
    .getOrCreate()
```

3. Define Kafka JSON Schema

- Purpose: Specifies the desired shape (schema) of the JSON messages which will be engaged in the Kafka topic.
- StructType(): The type used to represent a struct (an analog of dictionary or object) in the Spark type system.
- add(fieldName, DataType()): Adds a field in schema and its associated name and data type. The schema should conform to the JSON data that is generated by kafka_live_producer.py.

```
# Define the schema of incoming Kafka JSON
schema = StructType() \
    .add("app", StringType()) \
    .add("username", StringType()) \
    .add("review", StringType()) \
    .add("rating", StringType()) \
    .add("date", StringType()) \
    .add("fetched_at", StringType())
```

4. Read from Kafka Topic

- Purpose: Modifies Spark Structured Streaming to read data continuously from one of its topics in Kafka.
- spark.readStream It launches a streaming read.
- option("subscribe", "review_topic"): instructs Spark to subscribe to the topic of the Kafka with which it is named as review_topic.
- df_kafka will have Kafka-specific columns in each row with metadata (such as value (the actual message content)).

```
# Read from Kafka topic
df_kafka = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "review_topic") \
    .load()
```

5. Parse JSON Value

- Function: Retrieves the JSON string at the value column of the Kafka message and parses it based on the pre-determined schema.
- df_kafka.selectExpr("CAST(value AS STRING):" Converts the binary value column (which Kafka provides as a bytes) to a StringType.
- select(from_json(col("value"), schema).alias("data")): It is applied to the value column and uses the schema created above as a source of information. The JSON object became parsed and is alias named as data.
- .select("data.*"): Densifies the struct column of "data" (e.g., app, username, review, etc.) into actual columns and these columns can be accessed directly in the DataFrame.

```
# Parse JSON value
df_parsed = df_kafka.selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*")
```

6. Define UDF for LSTM Prediction

- Purpose: Specifies a User Defined Function (UDF) which will be used to encapsulate the logic for sentiment prediction so it could be used in a row-wise manner on a Spark DataFrame.
- predict_sentiment(text) function:
- As input, it takes a text (review content).
- unless text: return "neutral": If text is empty, it returns default of neutral.
- sequence = tokenizer.texts_to_sequences (text): Return a sequence of integers mapping the input text to the loaded tokenizer.
- padded = tf.keras.preprocessing.sequence.pad_sequences (sequence, maxlen=100): Trims or adds padding to sequences to a common length (100, in this case). Such is required because in most cases, neural networks require input of fixed size.
- prediction = model.predict(padded): The ready-to-use sequence is inserted into the downloaded LSTM model to receive the probability of the sentiment predictions.
- label_encoder.inverse_transform([np.argmax(prediction)])[0]:
- prediction: Returns the predicted sentiment class, as based on which the index of the largest probability in the prediction array is found.
- [0]: Pulls out the one-string label out of the resultant array.
- sentiment_udf = udf(predict_sentiment, StringType()): Property creates the UDF version of the Python predict_sentiment function.
- The first one is the very Python function.
- The second argument, StringType() is what the UDF will be returning (the sentiment label will be a string).

```
# Define UDF for LSTM prediction
def predict_sentiment(text):
    if not text:
        return "neutral"
    sequence = tokenizer.texts_to_sequences([text])
    padded = tf.keras.preprocessing.sequence.pad_sequences(sequence, maxlen=100)
    prediction = model.predict(padded)
```

```
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

sentiment_udf = udf(predict_sentiment, StringType())
```

7. Add Sentiment Prediction to DataFrame

- Purpose: uses sentiment_udf in the review column of the parsed DataFrame to produce a new column, named predicted_sentiment.
- df_parsed.withColumn(...): Appends a new column or substitutes the existing column in DataFrame.
- The name of the new column where the sentiment will be formed.

```
# Add sentiment prediction
df_with_sentiment = df_parsed.withColumn("predicted_sentiment",
sentiment_udf(col("review")))
```

8. Write to Console and Start Streaming Query

- Purpose: describes the format in which processed stream data (using the sentiment predictions) is to be sent and begins the streaming query.

```
# Write to console
query = df_with_sentiment.writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", False) \
    .start()

query.awaitTermination()
```

Run All Pipeline (run_all_pipeline.py)

The script is coordinating the work of Kafka producer, Spark consumer, and Streamlit dashboard that run simultaneously in different threads.

```
# run_all_pipeline.py

import subprocess
import threading
import time

# Script paths
KAFKA_PRODUCER_SCRIPT = "kafka_live_producer.py"
SPARK_CONSUMER_SCRIPT = "spark_kafka_consumer.py"
DASHBOARD_SCRIPT = "streamlit_dashboard.py"

def run_kafka_producer():
    print("      Launching Kafka Live Producer...")
    subprocess.run(["python", KAFKA_PRODUCER_SCRIPT])

def run_spark_consumer():
    print("      Launching Spark Kafka Consumer...")
    subprocess.run(["spark-submit", SPARK_CONSUMER_SCRIPT])

def run_dashboard():
    print("      Launching Streamlit Dashboard...")
    subprocess.run(["streamlit", "run", DASHBOARD_SCRIPT])

if __name__ == "__main__":
    print("      Starting Full Sentiment Analysis Pipeline...\n")

    kafka_thread = threading.Thread(target=run_kafka_producer)
    spark_thread = threading.Thread(target=run_spark_consumer)
    dashboard_thread = threading.Thread(target=run_dashboard)

    kafka_thread.start()
    time.sleep(5)

    spark_thread.start()
    time.sleep(10)

    dashboard_thread.start()

    kafka_thread.join()
    spark_thread.join()
    dashboard_thread.join()
```

5.0 Analysis & Results

The Real Time E-Wallet Sentiment Dashboard is capable of being constructed with Streamlit and provides a holistic platform through which the overall sentiment of the population and e-wallet applications the market can be understood. This section specifies the most important results obtained after the data processing, explains the visualizations applied such as the interactive visualizations, and outlines the insights that can be obtained and acted on.

5.1 Key Findings

Sentiment analysis of the e-wallet review reflected some important details of the perception and experience of the user:

- **Predominantly Positive Sentiment:** The general sentiment distribution shows that a considerable part of the reviews are positive. This implies a favourable reception and satisfaction of the people who use the chosen e-wallet applications in general. The numbers representing the reviews of the dashboard are clear on the positives prevailing numerically.
- **Presence of Neutral and Negative Feedback:** There are many positive reviews, but there is a great amount of neutral and negative ones as well. Neutral reviews usually mean lack of strong opinion or mixed feelings, and negative reviews show certain things which are wrong or areas that could be improved.
- **Dynamic Sentiment Trends:** The sentimental trend analysis dealt with the time factor, showing variations in the user sentiment. Such fluctuations might be associated with various reasons including app updates, advertisement campaigns or even service failures. It is vital to determine these time changes to find out how the events influence the perception of the audience.
- **Particular Problems That Are in the New Reviews:** Latest reviews section commonly gave micro level information about the sentiments. The typical dislikes in negative reviews that were classified across several types included app crashes, slow processing of transactions or unresponsive customer service, whereas the likes of the positive reviews mentioned being easy to use, cashback offers, and smooth transaction process.

5.2 Visualizations

The dashboard also utilizes multiple interactive visualizations to introduce the sentiment analysis results in an efficient way:

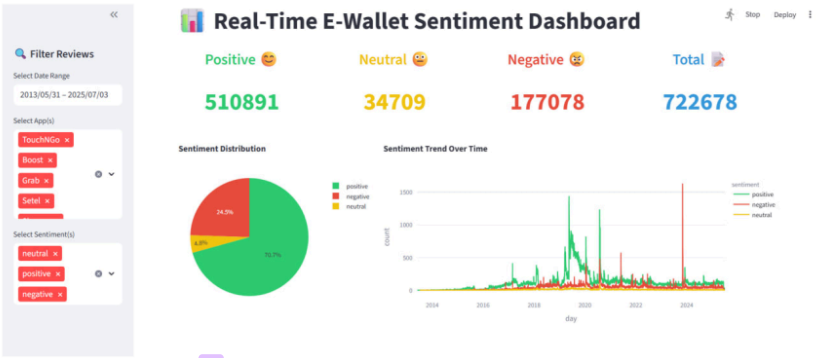


Figure 5.2.1 : Screenshot of dashboard interface 1

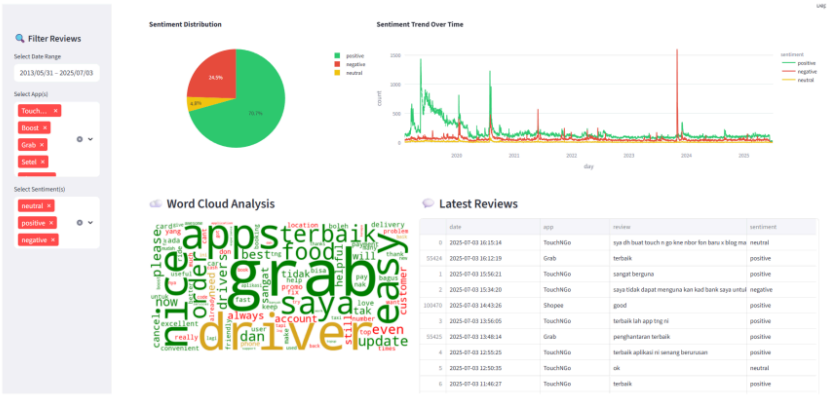


Figure 5.2.2 : Screenshot of dashboard interface 2

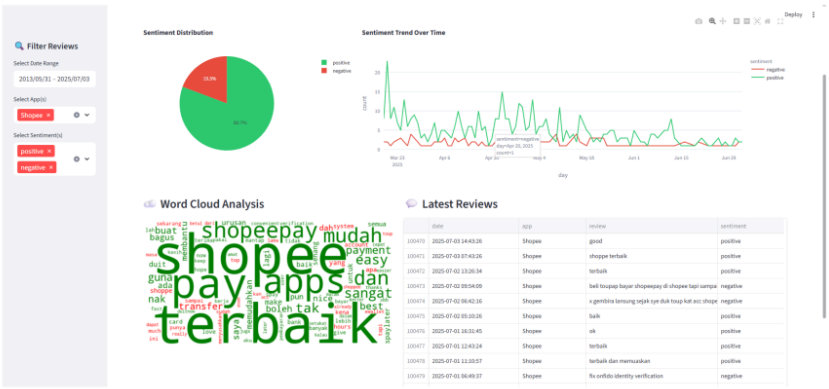


Figure 5.2.3 : Screenshot of dashboard for Shopee E-Wallet

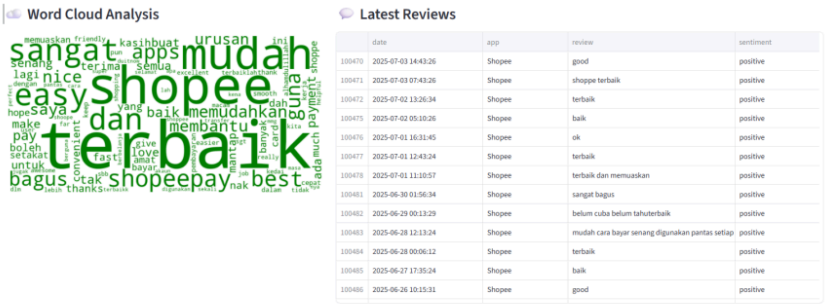
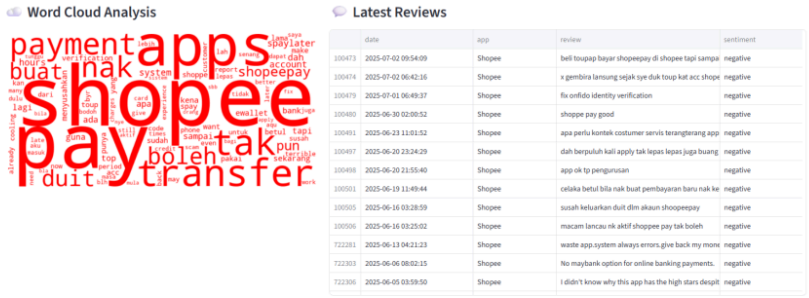


Figure 5.2.4 : Screenshot of positive reviews for Shopee

	date	app	review	sentiment
100507	2025-06-15 08:26:47	Shopee	baru join kasih star	neutral
100515	2025-06-12 14:24:05	Shopee	ok baik	neutral
100518	2025-06-11 12:12:25	Shopee	pergect	neutral
100524	2025-06-07 23:24:57	Shopee	kurmgmemuasaknsebb shopeepaysaya dah tk bole	neutral
722311	2025-06-01 15:14:10	Shopee	Had really love this apps, but can't transfer the mor	neutral
722325	2025-05-25 15:51:55	Shopee	Need to link more with shop to use shopee pay late	neutral
722326	2025-05-25 10:24:53	Shopee	orang nk bayar qr ko minta orang snuh tulis descrip	neutral
722355	2025-05-19 00:30:38	Shopee	why no more spaylater limit for me????	neutral
100550	2025-05-17 17:50:37	Shopee	lumayan	neutral
100561	2025-05-14 01:13:59	Shopee	kenapa proses duit nak masuk akaun bank lama sa	neutral
100568	2025-05-11 07:14:50	Shopee	masa susah macam ni jugak baru kau buat masalah	neutral
722418	2025-05-04 02:31:20	Shopee	lembab bila nak scan, meluat	neutral
100603	2025-04-28 18:22:16	Shopee	saya mau coba dulu	neutral

Figure 5.2.5 : Screenshot of neutral reviews for Shopee



	date	app	review	sentiment
100473	2025-07-02 09:54:09	Shopee	beli toupap bayar shopeepay di shopee tapi sampa	negative
100474	2025-07-02 06:42:16	Shopee	x gembira lansung sejak sye duk toup kat acc shope	negative
100479	2025-07-01 06:49:37	Shopee	fix onfido identity verification	negative
100480	2025-06-30 02:00:52	Shopee	shoppe pay good	negative
100491	2025-06-23 11:01:52	Shopee	apa perlu kontek costumer servis terangterang app	negative
100497	2025-06-20 23:24:29	Shopee	dah berpuluh kali apply tak lepas lepas juga buang	negative
100498	2025-06-20 21:55:40	Shopee	app ok tp pengurusan	negative
100501	2025-06-19 11:49:44	Shopee	celaka betul bila nak buat pembayaran baru nak ke	negative
100505	2025-06-16 03:28:59	Shopee	susah keluarkan duit dlm akaun shoopeepay	negative
100506	2025-06-16 03:25:02	Shopee	macam lancau nk aktif shoppee pay tak boleh	negative
722281	2025-06-13 04:21:23	Shopee	waste app.system always errors.give back my mone	negative
722303	2025-06-06 08:02:15	Shopee	No maybank option for online banking payments.	negative
722306	2025-06-05 03:59:50	Shopee	I didn't know why this app has the high stars despit	negative

Figure 5.2.6 : Screenshot of negative reviews for Shopee

- **Dashboard Overview and Interactivity**: The Streamlit is set in wide arrangement in order better to use space in detailed visualizations.

```
import streamlit as st
# ... other imports ...
st.set_page_config(page_title="E-Wallet Review Dashboard", layout="wide")
```

- The cleaned_reviews.csv is used as the main source of data and loaded up.

```
csv_path = 'cleaned_reviews.csv'
if not os.path.exists(csv_path):
    st.error("      Cleaned review file not found. Please check path.")
    st.stop()
df = pd.read_csv(csv_path, parse_dates=['date'])
```

- The sidebar offers interactive filters with which customers can select a required e-wallet application and use a custom date range. This makes it possible to analyse in detail particular applications or timelines.

```
# Sidebar filters
apps = df['app'].unique()
selected_apps = st.sidebar.multiselect("Select App(s)", apps, default=apps)
date_range = st.sidebar.date_input("Select Date Range",
    value=(df['date'].min().date(), df['date'].max().date())
)
# Apply filters
mask = (
    df['app'].isin(selected_apps) &
    (df['date'].dt.date >= date_range[0]) &
    (df['date'].dt.date <= date_range[1])
)
```

```
filtered_df = df[mask]
```

- **Important Sentiment Measures:** Three elements of the.metric component appear at the top of the dashboard and report the total number of positive, neutral, and negative reviews in the filtered data. They give real-time quantitative overview of sentiment distribution.

```
col1, col2, col3 = st.columns(3)
col1.metric("Positive", filtered_df[filtered_df['sentiment']=='positive'].shape[0])
col2.metric("Neutral", filtered_df[filtered_df['sentiment']=='neutral'].shape[0])
col3.metric("Negative", filtered_df[filtered_df['sentiment']=='negative'].shape[0])
```

- **Distribution of Sentiment Based on Types (Pie Chart):** The proportion of positive, neutral and negative sentiment categories is visualized using plotly.express.pie chart. This enables quick comprehension of the general sentiment.

```
fig1 = px.pie(filtered_df, names='sentiment', title='Sentiment Distribution')
st.plotly_chart(fig1, use_container_width=True)
```

- **Sentiment Trend Over Time (Line Chart):** The plotly.express.line chart is an expression of the daily tallies of positive, neutral and negative reviews. It is beneficial in terms of finding out the trends, surges or declines in attitude that may be related to certain events or product releases. Their counts of daily sentiments have been pre-processed.

```
filtered_df['day'] = filtered_df['date'].dt.date
fig2 = px.line(filtered_df.groupby(['day', 'sentiment']).size().reset_index(name='count'),
               x='day', y='count', color='sentiment', title='Sentiment Trend Over Time')
st.plotly_chart(fig2, use_container_width=True)
```

- **Latest Reviews (Dataframe):** The 20 latest reviews with their date, app name, the raw text they contained, and the sentiment in which they were classified are shown in a table via st.dataframe. This generates qualitative context and enables rapid verification of the classification or identification of the issues of sentiment.

```
st.subheader("      Latest Reviews")
st.dataframe(filtered_df.sort_values(by='date', ascending=False)[['date', 'app', 'review',
'sentiment']].head(20))
```

5.3 Insights

Some definite directions and steps that can be adopted by the e-wallet issuers and partners are made with the help of informative visualisations and insights on the dashboard:

- **Performance Monitoring:** A dashboard is an effective tool in monitoring the user happiness over time. By taking note of some of the key sentiment measures and their distribution, it will allow the stakeholders to gain a quick overview of the state of their e-wallet services.
- **Event Impact Analysis:** This is possible because the results of cause-and-effect are obtained based on the Sentiment Trend Over Time display. A sudden decline in the positive sentiment after an app update, say, may indicate a recently added bug or something, which was not very well-received among users, so that calls for an immediate investigation and response. Conversely, an increase in positive attitude after a marketing campaign ascertains the success of a campaign.
- **Specific areas of improvement:** Certain common areas of concern (e.g. "slow loading," "payment failures") could be discovered based on analyzing the latest reviews and clustering them by negative sentiment. This allows development groups to prioritise functional or bug-resolving changes that are most likely to remove user dissatisfaction.
- **Potential to Enhance Function:** Positive reviews may highlight which features or aspects of the service are the most popular or what can win the favour of the customers particularly. The findings can become a guide to future attempts at development, with a focus on developing or improving on functionalities that have been successful.
- **Competitive Analysis (in case there is data on a range of apps):** Filtering by specific e-wallet applications, a company can consider how good it is sentimentally against its counterparts (in case there is data on a range of apps), and in which situations it surpasses or lacks.
- **Data Driven Decision Making:** Ultimately the dashboard takes raw, unstructured data on reviews and turns them into the form of easily interpreted and consumable insight. This empowers marketing teams, customer services agents and product managers to make informed decisions that will lead to improved user experience and retention.

6.0 Optimization & Comparison

6.1 Model or Architecture Improvements

In this section, the description of the improvements and optimizations made to the sentiment classification models as well as to the underlying real time data processing architecture are provided with the purpose of improvement of its performance, accuracy, and efficiency.

6.2 Sentiment Model Enhancement

In our project, two different techniques of sentiment classification have been applied, a Multinomial Naive Bayes (MNB) classifier, and a neural network based on Long Short-Term Memory (LSTM). The models follow a certain preprocessing and training and can be improved.

6.2.1 Model Incrementalities On Naive Bayes:

TF-IDF vectorization is used as the Naive Bayes model matches text into numerical features by setting the value of max_features=5000. Reviews are divided into test sets and training sets (80/20 with random_state= 42).

1. **Beyond common TF-IDF Feature engineering:**

- **N-grams:** Where TF-IDF captures the importance of words, one could be interested in sentences and common phrases that have the sentiment, such as not good or very happy through the inclusion of n-grams (two words, three words, etc.). It can be done, by modifying the parameter ngram_range in TfidfVectorizer.
- **Features based on the lexicon:** Incorporation of sentiment lexicons (e.g., SentiWordNet, VADER) are possible which might add features into the model specially induced on tweaking the elements of informal text or slang that usually prevails in reviews.

2. **Model Ensemble:** Rather than leaving all the predictions to Naive Bayes, this approach could also be used to merge such predictions with others that have made more accurate predictions, in models that are simpler (e.g., Logistic Regression, that I had initially proposed to start with in model_training.ipynb before focusing on Naive Bayes and LSTM).

6.2.2 Improvements of LSTM Model:

The model loaded is LSTM where a pre-trained tokenizer.pkl and labelencyoder.pkl is used. The preprocessing operation encompasses tokenization and pad_sequences having max len=100.

1. **Hyperparameter Tuning:** Our LSTM has Embedding, LSTM(128, return_sequences=True), Dropout(0.2), LSTM(64), Dropout(0.2), and lastly a Dense softmax layer which is compiled using Adam, and the categorical_crossentropy. Additional adjustment may include:

- **Maxlen:** There may be some potential in trying out other values of pad_sequences.maxlen. The shorter maxlen value then may remove the computational overheads but can cut off long and informative reviews and the longer value may add the padding that was unwanted but context is not being ignored.
- **LSTM Units:** Changing the amount of units in the LSTM layers (e.g. 128 and 6LSTM layers) and the amount of LSTM layers.
- **Dropout Rates:** Adjusting dropout rates so as to be able to avoid overfitting and still maintain model capacity.
- **Batch Size and Epochs:** choosing them well in training may greatly improve the training length and convergence.

2. **Another Method for Embedding :**

- **Pre trained Word Embeddings:** Use pre trained word like good word so embeddings can learn much more rich semantic relationships than the tokenizer/embedding layer could learn during training, and can significantly improve model performance, especially

with sparse training data, e.g. Word2Vec, GloVe, or FastText (especially those trained on Bahasa Melayu corpora).

- **Contextual Embeddings (Transfer Learning):** To achieve the best performance, one should consider fine tuning of pre trained transformer based models like BERT (Bidirectional Encoder Representations from Transformers) RoBERTa. Such models have a superior comprehension of the word context and are able to work out sentiments on an advanced level. Even though they are more resource heavy, they also provide a considerable performance benefit.

3. **The actual model inference:**

- To convert the LSTM model (sentiment_lstm_model.h5) so that it could run in real time on Spark, we can use such techniques as model quantization. This makes the model less accurate (e.g. float32 to int8) thus making the model smaller and faster to predict without significant differences to actuarial accuracy, which is important to a high throughput streaming pipeline.

6.3 Architecture Improvement

Our project uses a powerful real time architecture using Apache Kafka to transport data and the Apache Spark to transfer stream processing and prediction of sentiment and Streamlit for visualization. It is possible to optimize the scalability, fault tolerance and latency of this pipeline.

6.3.1 Optimization Apache Kafka:

- **Topic Partitioning:** Our data flow is about the review_topic. In order to enhance throughput and parallelism, scale more partitions of review_topic in Kafka. The more the partitions, the more the number of Spark executors that can read messages simultaneously.
- **Producer Tuning:** The scripts kafka_live_producer.py and kafka_producer_reviews.py will send a message to Kafka. Set up producer settings such as linger.ms (set to time out to send your accumulated messages) and batch.size (sets the largest size of a batch) to

minimize overhead and improve network efficiency. Moreover, the fault tolerance in network related issues can be improved by configuring the retries.

- **Replication factor:** In production, make sure that the replication factor of Kafka topics is more than 1 which will help us not lose the data in case of broker failures.

6.3.2 Apache Spark streaming Improvements:

1. **Resource Allocation:** Our run_all_pipeline.py is invoked with the spark-submit command that starts up our Spark consumer. Set parameters of Spark resource allocation:

- **spark.executor.memory:** Provide enough memory to executors such that spark does not fail because of OOM issues.
- **spark.executor.cores:** Manage the amount of CPU cores the executors can use.
- **spark.executor.instances:** This argument lets us change how many executors are used depending on the size of the cluster and the amount of processing that has to be completed.
- **spark.default.parallelism:** Set it to the parameter that regards the number of Spark cores and the number of partitions in Kafka topic so that the task load can be balanced.

2. **Micro Batching Interval:** Our Spark streaming application stream processes data in terms of micro batches. In our Spark consumer (spark_kafka_consumer.py) experiment with two factors: the spark.streaming.kafka.maxRatePerPartition config parameter and the streamingContext.awaitTerminationOrTimeout method to strike a balance between latency and throughput. The shorter the batch interval the lower the latency but the higher the overhead whereas a longer batch interval brings more latency but can also better overall throughput.

3. **UDF Optimization:** This UDF plays a vital role in creating a real time inference such as predict_sentiment. For Spark UDFs are performance bottlenecks, so are serialization/deserialization overhead, although they still can be used still.

- Pandas UDFs (Vectorized UDFs) may be used, when possible, with our TensorFlow model loading, and can dramatically increase performance by running a batch of data at a time instead of row wise.
- Instead, when the model inference is very heavy consider deploying the model as a microservice in itself and having Spark call its service, which increases the architectural complexity.

4. **Checkpointing:** To be fault tolerant and keep the data recovered, so should be the Sparks checkpointing in the streaming context (spark.readStream) that does not require starting the data reprocessing at the beginning in case of application failures.

5. **Output Sink:** We are currently using Spark consumer that produces CSV files. In real time dashboarding change over to more appropriate sinks:

- **Elasticsearch:** Pushing the processed data directly to the Elasticsearch would allow it to immediately index the data and perform real-time queries to your Streamlit dashboard, as it was stated in our project outline.
- **Apache Druid:** A good alternative to real time analytics and fast aggregations also to run the visualizations in Streamlit.

C. Fault Tolerance and Overall Scalability:

- **Horizontal Scaling:** Infrastructure Horizontal scale of the architecture can be achieved simply by increasing the number of Kafka brokers and Spark worker nodes to account more data ingestion and processing.
- **Data persistence:** Upgrading CSV files into a powerful and distributed NoSQL database such as Elasticsearch along with a real time analytics database such as Druid can guarantee data persistence and high availability of the dashboard. This eradicates the dependency on the local file systems and increases the durability of the complete system.

Conclusion for Improvement:

Through these enhancements, real time sentiment analysis pipeline could attain improved accuracy, less latency, and be more robust, which are among the many demands of a high performance data processing system.

7.0 Conclusion & Future Work

7.1 Conclusion

This initiative actually proposed and realized a backward real time estimation pipeline on the sentiments of Malaysian e-wallet evaluations by using an impressive alliance of Apache Spark and Kafka. We have shown the entire pipeline of data retrieval, complete preprocessing of unstructured text based data, the creation and publishing of sentiment understanding models, and ultimately, a way to see such results in a real time interactive manner via Streamlit dashboard view.

7.1.1 Some of the main accomplishments are as follows:

- **Real time Data Ingestion:** Real time data ingestion of streaming live e-wallet reviews in Google Play Store to Apache Kafka through custom Python producers (kafka_live_producer.py) which allows live streaming of data.
- **Powerful Data Preprocessing:** Applied an extensive preprocessing pipeline (preprocessing.ipynb) with the aim to detecting languages, cleaning (removing URLs, punctuation, numbers, special symbols), performing stopword removal and stemming, in fact specific to Bahasa Melayu utterances.
- **Sentiment Model Development:** Trained and tested two different models of sentiment classification, a Multinomial Naive Bayes model on features based on TF-IDF and a deep learning LSTM model, and evidenced the uses of not only traditional machine learning but also deep learning in predicting sentiment (model_training.ipynb).
- **Scalable Real time Processing:** Ran Apache Spark Streaming (spark_kafka_consumer.py) to process large amounts of data in a distributed system that reads real time data into Kafka, and sends it through the trained sentiment model (LSTM) to demonstrate Spark shines as a unified analytics engine to perform streaming applications.
- **Intelligent Visualization:** Drafted an interactive Streamlit dashboard (streamlit_dashboard.py) in which the most important sentiment measures, distribution,

and time series could be observed, enabling the users (stakeholders) to develop real-time and tactical insight into the attitude of the masses concerning e-wallet applications.

- **Pipeline Integration:** Carried out the effective combination of an assortment of Apache solutions (Kafka, Spark) and different tools (Streamlit, Pandas, Scikit-learn, TensorFlow / Keras) into a practical and intact real-time analytics pipe, demonstrating key data engineering and data science abilities.

The value of real-time sentiment analysis in informing dynamic public opinion within a business environment is immeasurable and therefore, by developing this project, businesses will be able to preempt user responses and feedback, develop woodholes on emergent issues and be able to make data based decisions that will not only inform, but also influence product development, product outreach and consumer satisfaction.

7.2 Future Work

In order to improve the capacity and strength of the pipeline of this real-time sentiment analysis, the following options are presented to be worked upon:

1. **Improved and Advanced Sentiment Models:**

- **Contextual Embeddings:** Consider training and fine-tuning the latest transformer based pre trained models (e.g., BERT, XLM-RoBERTa or even Bahasa Melayu BERT variants) with an eye towards gaining more specific understanding of sentiment, including culturally specific multiple meanings, and also more specific understanding of idiomatic expressions, especially in a social media text context.
- **Aspect-Based Sentiment Analysis:** In addition to having a general positive/neutral/negative label, find out what people think about the e-wallet apps based on the different features/aspects of it (e.g. how fast the payments go through, how does the app look, how good is the customer support)? This would also give more detailed actionable information.

- **Ensemble Modeling:** Explore the state of art ensemble methods that use a collection of various models to come up with better predictive accuracy and are generally consistent.

2. **Real-Time Data storage and persistence:**

- **Action Elasticsearch Integration:** Refactor the Spark streaming so it writes to Elasticsearch (rather than writing the local CSV files (merge_csv_output.py) and then later pushing to Elasticsearch). The result would allow an actual real time indexing and search taking advantage of the live, scalable datastore as intended by the streamlit dashboard internal organization.
- **Apache Druid:** Alternatively, look into the possibility of Apache Druid serving as the core of real-time analytics due to low latency queries and aggregation, this would best be used to drive real time dashboards and real-time reports.

3. **Improved Scalability and fault Tolerance:**

- **Spark Cluster Tuning:** Tune spark cluster settings more thoroughly (e.g. spark.executor.memory, spark.executor.cores, spark.default.parallelism) in order to maximize resource usage and rates of throughput with bigger volumes of data.
- **Kafka Cluster Management:** Set up a Kafka cluster monitoring and auto scaling to maintain an appropriate readiness to fluctuating message rates so that there would be no data backlog.
- **Resilience Testing:** Take the pipeline through thorough fault injection tests to determine correctness of recovery mechanisms of the pipeline and how it reacts when a component fails.

4. **On a larger scale, these sources of data are globally available and can be multimodal:**

- **Increased Data Collection:** Consolidate information provided by more Malaysian relevant social media platforms (e.g., X/Twitter, comments on Facebook, local forums,

news articles), to give a broader picture of how people comment on different social media.

- **Multimodal Sentiment Analysis:** Consider adding non textual inputs (e.g. emojis as explicit features, content of images, in the case social media posts) to complement the sentiment analysis as the use of emojis is common in the reviews on mobile apps.

5. **Advanced Dashboard Capabilities and Warning:**

- **Predictive Analytics:** Build predictive models into the dashboard and show the forecast of the future trend of sentiment or problems that may occur, using historical data.
- **Anomaly Detection and Alerting:** This can be done by setting up automatic alerts (e.g. via a timely email or Slack message) that must be triggered in case of relevant anomalies or critical changes in negative sentiment, to then be able to take action quickly in case of an emergency.
- **Interactive Drill downs:** Provide more drilling down features on the Streamlit dashboard, where the users can drill down based on sentiment to particular apps, features, or even single keywords.

This by exploring these directions will bring the project to become an even more mature, heavy, and effective real-time sentiment analysis system used to project deeper and more effective operational intelligence.

8.0 References

Apache Software Foundation. (2023). Apache Kafka documentation. https://kafka.apache.org/documentation/

Apache Software Foundation. (2023). Apache Spark: Unified engine for large-scale data analytics. https://spark.apache.org/docs/latest/

TensorFlow Team. (2023). TensorFlow API documentation. https://www.tensorflow.org/api_docs

TensorFlow. (n.d.). TensorFlow: An end-to-end open-source machine learning platform. https://www.tensorflow.org/

Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, *12*, 2825–2830. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Streamlit Team. (2023). Streamlit documentation. https://docs.streamlit.io/

VanderPlas, J. (2016). Python data science handbook. O'Reilly Media. https://jakevdp.github.io/PythonDataScienceHandbook/

9.0 Appendix

All codes and resources can be found on link below

https://github.com/drshahizan/HPDP/tree/main/2425/project/p2/DataDrillers

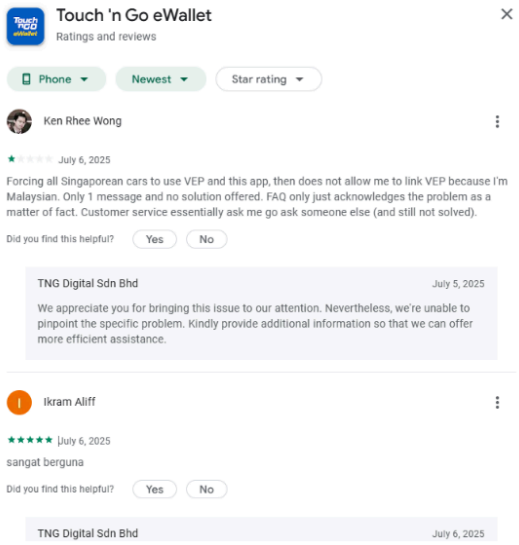


Figure 9.0.1 : Examples of Touch 'n Go eWallet reviews on Google Play Store

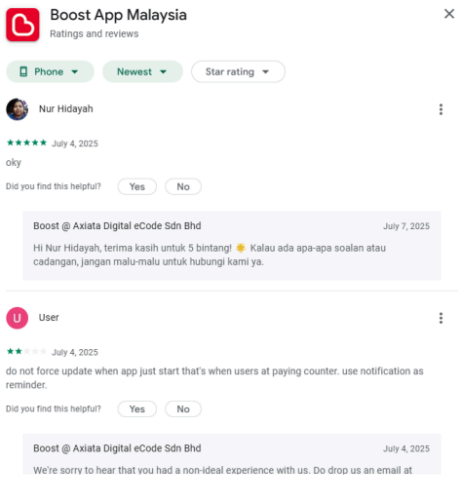


Figure 9.0.2 : Examples of Boost reviews on Google Play Store

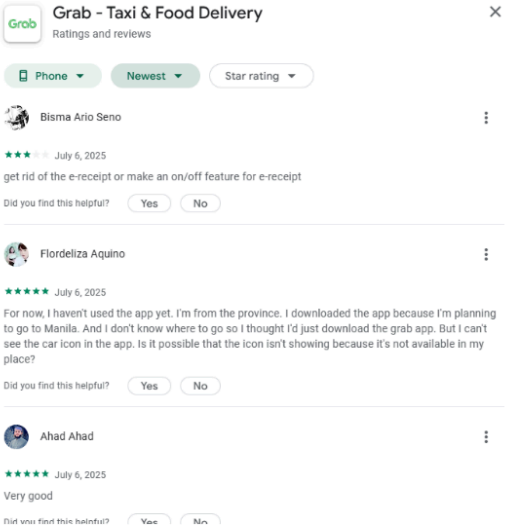


Figure 9.0.3 : Examples of Grab reviews on Google Play Store

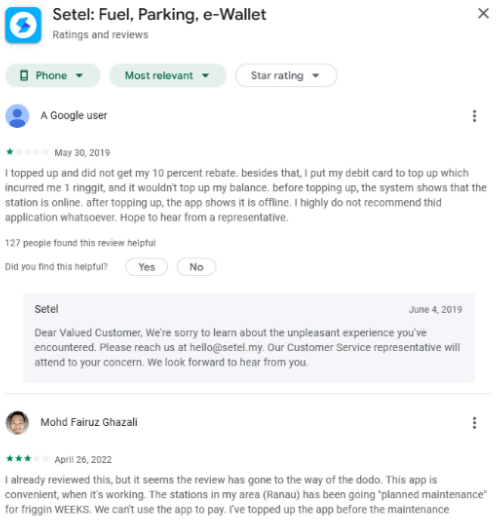


Figure 9.0.4 : Examples of Setel reviews on Google Play Store

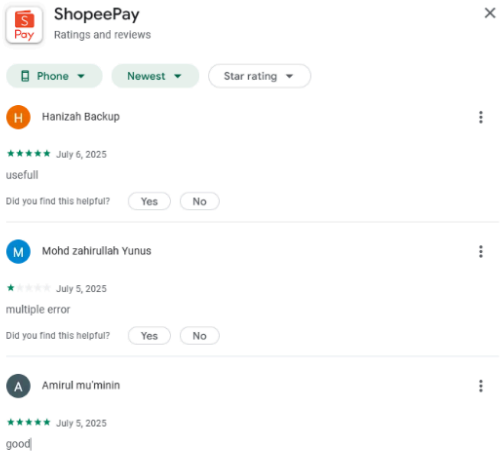


Figure 9.0.5 : Examples of Setel reviews on Google Play Store

PROJ2_DataDrillers

ORIGINALITY REPORT

7%	6%	2%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	editor.ploomber.io Internet Source	1%
2	Submitted to Sydney Polytechnic Institute Student Paper	<1%
3	www.coursehero.com Internet Source	<1%
4	Submitted to University of Hertfordshire Student Paper	<1%
5	www.chaosgenius.io Internet Source	<1%
6	Submitted to CSU, San Jose State University Student Paper	<1%
7	medium.com Internet Source	<1%
8	www.getorchestra.io Internet Source	<1%
9	Submitted to University of Adelaide Student Paper	<1%
10	dokumen.pub Internet Source	<1%
11	www.ips.edu.in Internet Source	<1%
12	origin.geeksforgeeks.org Internet Source	<1%
13	www.kdnuggets.com Internet Source	<1%

14	ichi.pro Internet Source	<1%
15	Submitted to City University of Hong Kong Student Paper	<1%
16	Submitted to University of North Texas Student Paper	<1%
17	ia903401.us.archive.org Internet Source	<1%
18	www.uky.edu Internet Source	<1%
19	stackoverflow.com Internet Source	<1%
20	staging-jmir.jmir.org Internet Source	<1%
21	Nur Nasuha Daud, Siti Hafizah Ab Hamid, Muntadher Saadoon, Chempaka Seri, Zati Hakim Azizul Hasan, Nor Badrul Anuar. "Self-Configured Framework for scalable link prediction in twitter: Towards autonomous spark framework", Knowledge-Based Systems, 2022 Publication	<1%
22	appwrite.io Internet Source	<1%
23	community.databricks.com Internet Source	<1%
24	library.samdu.uz Internet Source	<1%
25	web.wpi.edu Internet Source	<1%
26	www.geeksforgeeks.org Internet Source	<1%

27	www.ijtrd.com Internet Source	<1 %
28	www.mssl.ucl.ac.uk Internet Source	<1 %
29	www.mssqltips.com Internet Source	<1 %
30	www.researchgate.net Internet Source	<1 %
31	Submitted to Mepco Schlenk Engineering college Student Paper	<1 %
32	aclanthology.org Internet Source	<1 %
33	globaljournals.org Internet Source	<1 %
34	Satyajit Chakrabarti, Ashiq A. Sakib, Souti Chattopadhyay, Sanghamitra Poddar, Anupam Bhattacharya, Malay Gangopadhyaya. "Interdisciplinary Research in Technology and Management", CRC Press, 2024 Publication	<1 %
35	"Proceedings of Ninth International Congress on Information and Communication Technology", Springer Science and Business Media LLC, 2024 Publication	<1 %
36	version.aalto.fi Internet Source	<1 %
37	ela.kpi.ua Internet Source	<1 %
38	speakerdeck.com Internet Source	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		