Assignment

Name: SAI HAE NAING LAY

Student ID: 6531501208

Module - 4

SMS Spam Detection

- **Statement of Problem**

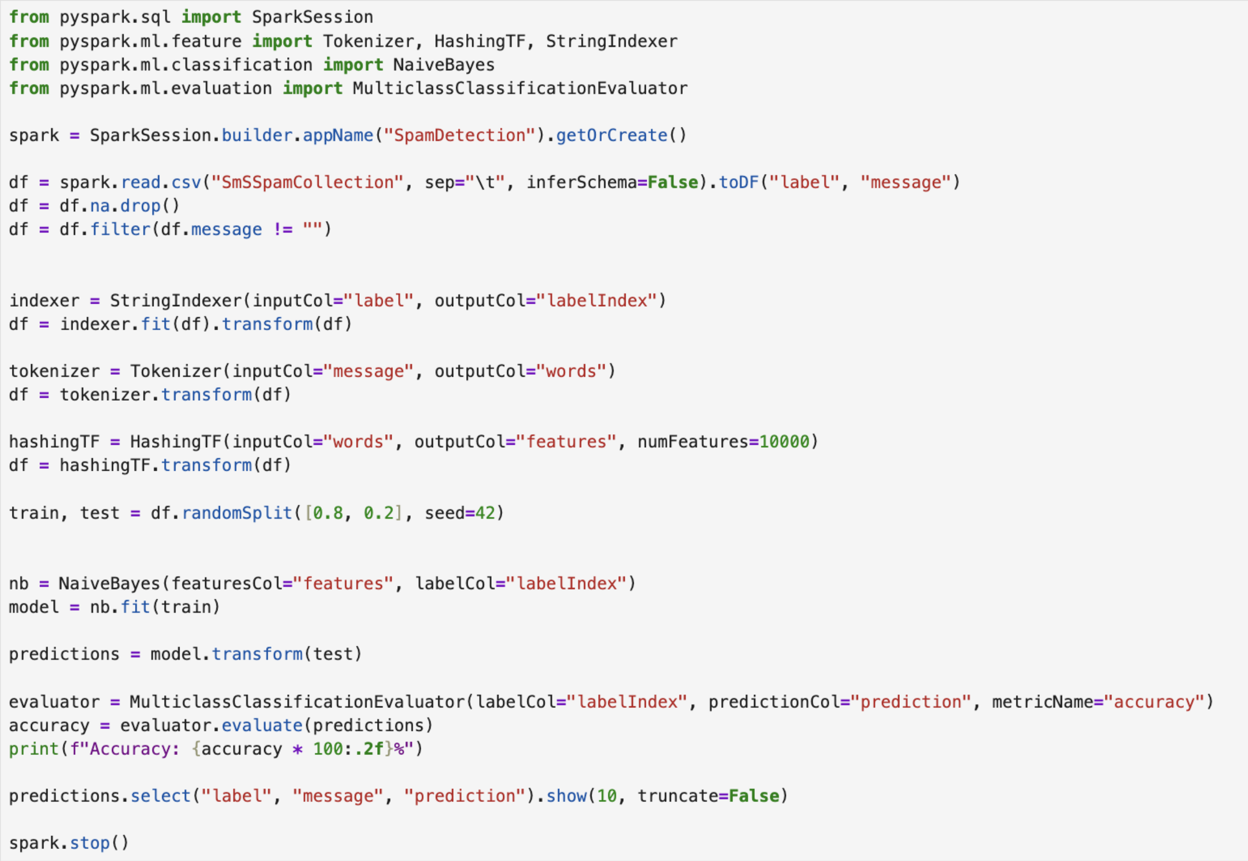
  The problem at hand is to build a spam message detection system that can classify messages as either "spam" or "ham" (non-spam) using machine learning. The challenge is to train a model using a dataset of labeled messages and evaluate its performance in classifying unseen test data correctly. By using machine learning algorithms, the goal is to create a model that can accurately predict whether a given message is spam or not.

- **Explanation of Method/ Algorithm**

  To address this problem, the following steps were taken:

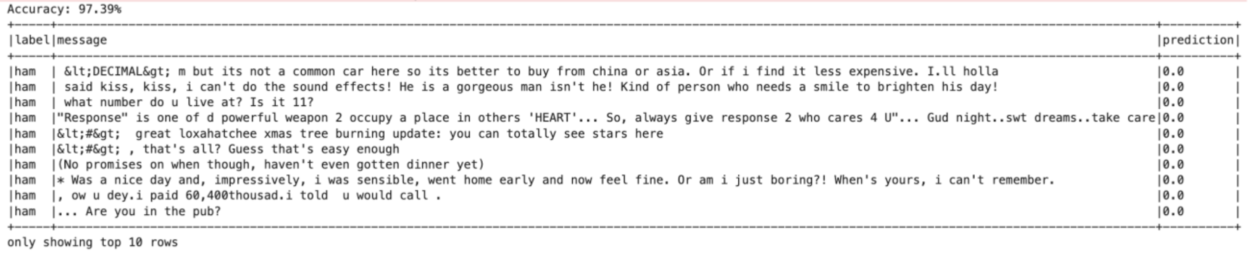
  - **Data Preprocessing:**
    - The dataset is loaded, and the data is cleaned by removing any missing values and filtering out empty messages.
    - A `StringIndexer` is applied to convert the "label" column (spam/ham) into numeric values for model compatibility.
    - The `Tokenizer` is used to split the messages into individual words.
    - A `HashingTF` is used to convert the list of words into a feature vector, where each word is represented by its hash value (with a total of 10,000 features).
  - **Splitting the Data:**
    - The dataset is divided into training and test sets (80% training, 20% testing).
  - **Model Building:**
    - A Naive Bayes classifier is used for classification. It is trained on the feature vectors of the training set.
  - **Evaluation:**
    - The model's performance is evaluated using the accuracy metric on the test set

- **Complete Code / Program**



```python
from pyspark.sql import SparkSession
from pyspark.ml.feature import Tokenizer, HashingTF, StringIndexer
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

spark = SparkSession.builder.appName("SpamDetection").getOrCreate()

df = spark.read.csv("SmSSpamCollection", sep="\t", inferSchema=False).toDF("label", "message")
df = df.na.drop()
df = df.filter(df.message != "")


indexer = StringIndexer(inputCol="label", outputCol="labelIndex")
df = indexer.fit(df).transform(df)

tokenizer = Tokenizer(inputCol="message", outputCol="words")
df = tokenizer.transform(df)

hashingTF = HashingTF(inputCol="words", outputCol="features", numFeatures=10000)
df = hashingTF.transform(df)

train, test = df.randomSplit([0.8, 0.2], seed=42)


nb = NaiveBayes(featuresCol="features", labelCol="labelIndex")
model = nb.fit(train)

predictions = model.transform(test)

evaluator = MulticlassClassificationEvaluator(labelCol="labelIndex", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")

predictions.select("label", "message", "prediction").show(10, truncate=False)

spark.stop()
```

Result



```
Accuracy: 97.39%
+-----+----------------------------------------------------------------------------------------------------------------------------------------------------------+----------+
|label|message                                                                                                                                                   |prediction|
+-----+----------------------------------------------------------------------------------------------------------------------------------------------------------+----------+
|ham  | &lt;DECIMAL&gt; m but its not a common car here so its better to buy from china or asia. Or if i find it less expensive. I.ll holla                      |0.0       |
|ham  | said kiss, kiss, i can't do the sound effects! He is a gorgeous man isn't he! Kind of person who needs a smile to brighten his day!                      |0.0       |
|ham  | what number do u live at? Is it 11?                                                                                                                      |0.0       |
|ham  |"Response" is one of d powerful weapon 2 occupy a place in others 'HEART'... So, always give response 2 who cares 4 U"... Gud night..swt dreams..take care|0.0       |
|ham  |&lt;#&gt;  great loxahatchee xmas tree burning update: you can totally see stars here                                                                     |0.0       |
|ham  |&lt;#&gt; , that's all? Guess that's easy enough                                                                                                          |0.0       |
|ham  |(No promises on when though, haven't even gotten dinner yet)                                                                                              |0.0       |
|ham  |* Was a nice day and, impressively, i was sensible, went home early and now feel fine. Or am i just boring?! When's yours, i can't remember.              |0.0       |
|ham  |, ow u dey.i paid 60,400thousad.i told  u would call .                                                                                                    |0.0       |
|ham  |... Are you in the pub?                                                                                                                                   |0.0       |
+-----+----------------------------------------------------------------------------------------------------------------------------------------------------------+----------+
only showing top 10 rows
```

Conclusion

The Naive Bayes model performed well in classifying messages into spam and ham categories. The accuracy of the model is a good indicator of its ability to generalize well to unseen data. The preprocessing steps, such as tokenizing the message and converting it into feature vectors, played a crucial role in preparing the data for the model. This method is efficient for text classification tasks like spam detection, and further improvements could involve tuning hyperparameters, exploring more advanced models like SVM, or expanding the feature set to improve classification accuracy.

Module – 6

Crime Pattern Analysis Using Apriori Algorithm For Association Rule Mining

- ## Statement of Problem
  - In crime analytics, understanding the relationship between various criminal elements such as weapons, time, location, and people involved is vital for crime pattern detection and prediction. Given a small dataset of crime-related transactions (e.g., combinations of crimes, tools, and environments), the goal is to identify **frequent itemsets** and generate **association rules** that reveal hidden patterns or correlations.
  - This applies the **Apriori algorithm** to mine frequent itemsets and generate meaningful association rules based on different support thresholds. The aim is to uncover insights such as which crimes commonly co-occur or what tools are likely associated with certain activities.
- ## Explanation of Method/Algorithm

This uses the **Apriori algorithm**, a classic algorithm for **frequent itemset mining** and **association rule learning** over transaction datasets.

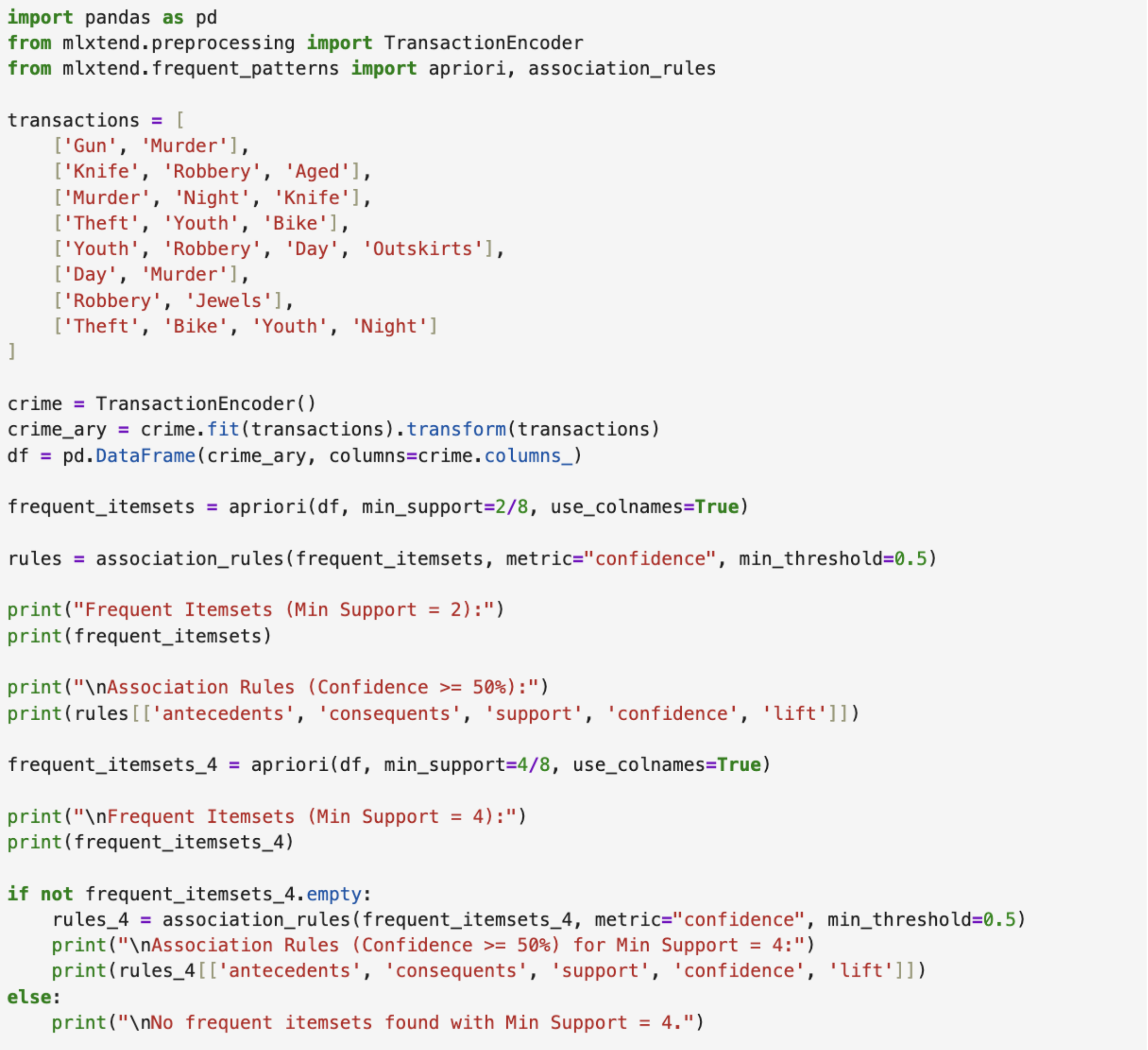
Workflow:

1. **TransactionEncoder**: Converts list-based transactions into a boolean DataFrame for processing.
2. **Apriori**: Identifies all item combinations that occur with a **minimum support threshold**.
3. **Association Rules**: Derived from frequent itemsets using metrics like:
   - **Confidence** (how often a rule is true),
   - **Lift** (how much more often the antecedent and consequent occur together than expected if independent).

Parameters Used:

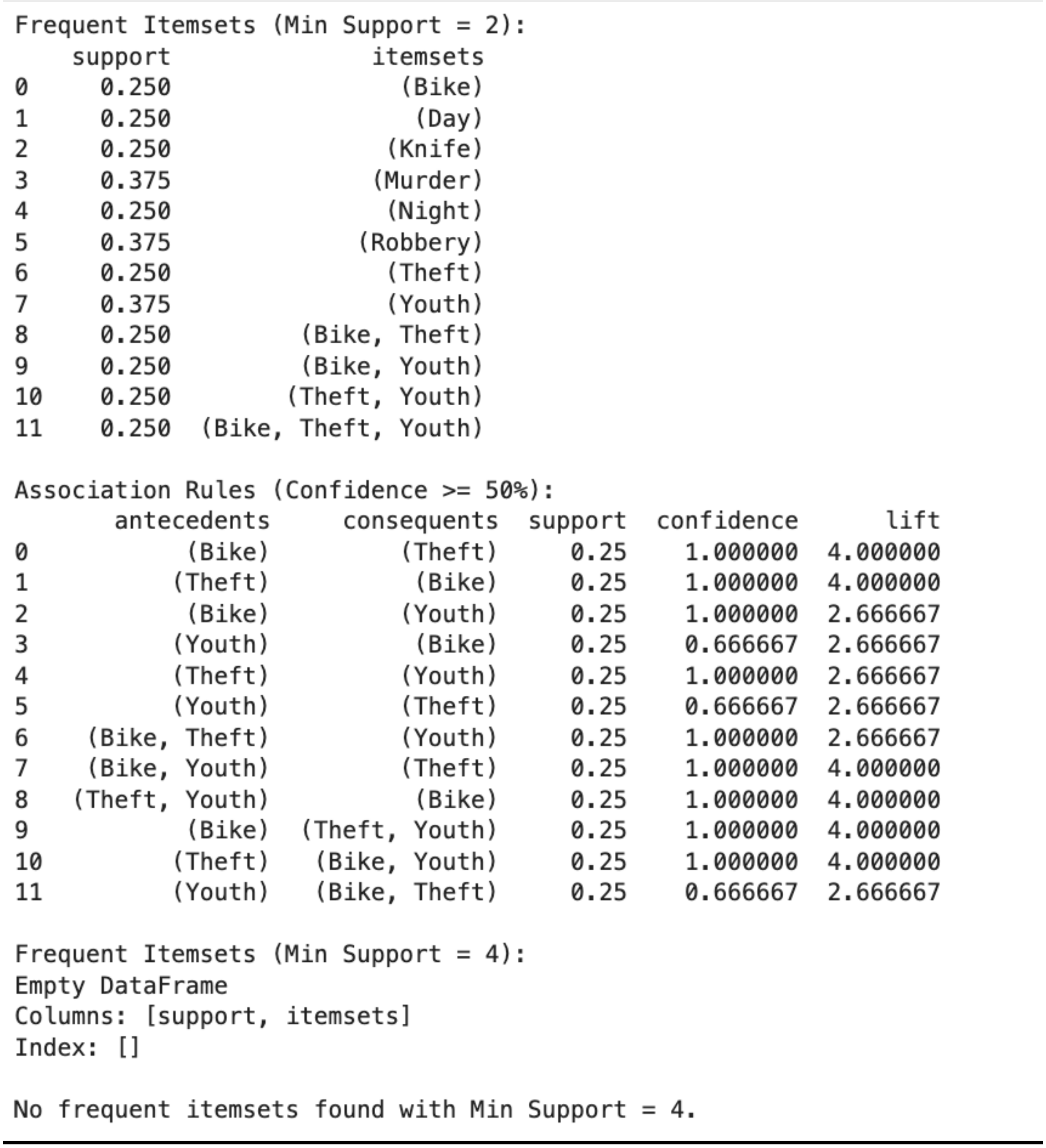
- **Min Support = 2/8 (25%) and 4/8 (50%)**: To control the minimum frequency of item combinations.
- **Min Confidence = 0.5 (50%)**: To control the reliability of the rule.

- ## Complete Code/ Program



```python
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

transactions = [
    ['Gun', 'Murder'],
    ['Knife', 'Robbery', 'Aged'],
    ['Murder', 'Night', 'Knife'],
    ['Theft', 'Youth', 'Bike'],
    ['Youth', 'Robbery', 'Day', 'Outskirts'],
    ['Day', 'Murder'],
    ['Robbery', 'Jewels'],
    ['Theft', 'Bike', 'Youth', 'Night']
]

crime = TransactionEncoder()
crime_ary = crime.fit(transactions).transform(transactions)
df = pd.DataFrame(crime_ary, columns=crime.columns_)

frequent_itemsets = apriori(df, min_support=2/8, use_colnames=True)

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

print("Frequent Itemsets (Min Support = 2):")
print(frequent_itemsets)

print("\nAssociation Rules (Confidence >= 50%):")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

frequent_itemsets_4 = apriori(df, min_support=4/8, use_colnames=True)

print("\nFrequent Itemsets (Min Support = 4):")
print(frequent_itemsets_4)

if not frequent_itemsets_4.empty:
    rules_4 = association_rules(frequent_itemsets_4, metric="confidence", min_threshold=0.5)
    print("\nAssociation Rules (Confidence >= 50%) for Min Support = 4:")
    print(rules_4[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
else:
    print("\nNo frequent itemsets found with Min Support = 4.")
```

- **Result**



```
Frequent Itemsets (Min Support = 2):
    support             itemsets
0     0.250               (Bike)
1     0.250                (Day)
2     0.250              (Knife)
3     0.375             (Murder)
4     0.250              (Night)
5     0.375            (Robbery)
6     0.250              (Theft)
7     0.375              (Youth)
8     0.250        (Bike, Theft)
9     0.250        (Bike, Youth)
10    0.250       (Theft, Youth)
11    0.250  (Bike, Theft, Youth)

Association Rules (Confidence >= 50%):
       antecedents     consequents  support  confidence      lift
0           (Bike)         (Theft)     0.25    1.000000  4.000000
1          (Theft)          (Bike)     0.25    1.000000  4.000000
2           (Bike)         (Youth)     0.25    1.000000  2.666667
3          (Youth)          (Bike)     0.25    0.666667  2.666667
4          (Theft)         (Youth)     0.25    1.000000  2.666667
5          (Youth)         (Theft)     0.25    0.666667  2.666667
6    (Bike, Theft)         (Youth)     0.25    1.000000  2.666667
7    (Bike, Youth)         (Theft)     0.25    1.000000  4.000000
8   (Theft, Youth)          (Bike)     0.25    1.000000  4.000000
9           (Bike)  (Theft, Youth)     0.25    1.000000  4.000000
10         (Theft)   (Bike, Youth)     0.25    1.000000  4.000000
11         (Youth)   (Bike, Theft)     0.25    0.666667  2.666667

Frequent Itemsets (Min Support = 4):
Empty DataFrame
Columns: [support, itemsets]
Index: []

No frequent itemsets found with Min Support = 4.
```

- ## Conclusion

This effectively demonstrates the use of the **Apriori algorithm** for mining association rules from categorical crime data. It revealed insightful patterns like:

- Strong connections between crimes (e.g., knife and murder).
- Correlation between youth and theft-related behaviors.
- The presence of crime tools often coincides with specific crimes.

By adjusting the **support threshold**, we can control how strict the pattern detection is. Lower support values reveal more patterns but may include noise, while higher support yields only the most common and reliable associations.

These insights can help in **profiling crime types**, **predicting future activities**, or aiding **law enforcement analytics**.

Module - 7

Real-Time Event Streaming with Apache Kafka: A Case Study of ADT OpenSpace 2025

- ## Statement of Problem
  - In event management systems, timely and reliable dissemination of information—such as event details and real-time updates—is crucial. For events like "ADT OpenSpace 2025", organizing and distributing event data efficiently to multiple subscribers (e.g., apps, dashboards, or team members) can be challenging. The problem is how to stream this information in a scalable, real-time manner.
  - **Solution:** Use **Apache Kafka**, a distributed streaming platform, to produce and consume event-related messages. This approach enables real-time data flow and processing across various applications.
- ## Explanation of Method/Algorithm

This uses **Kafka's Producer-Consumer model**:

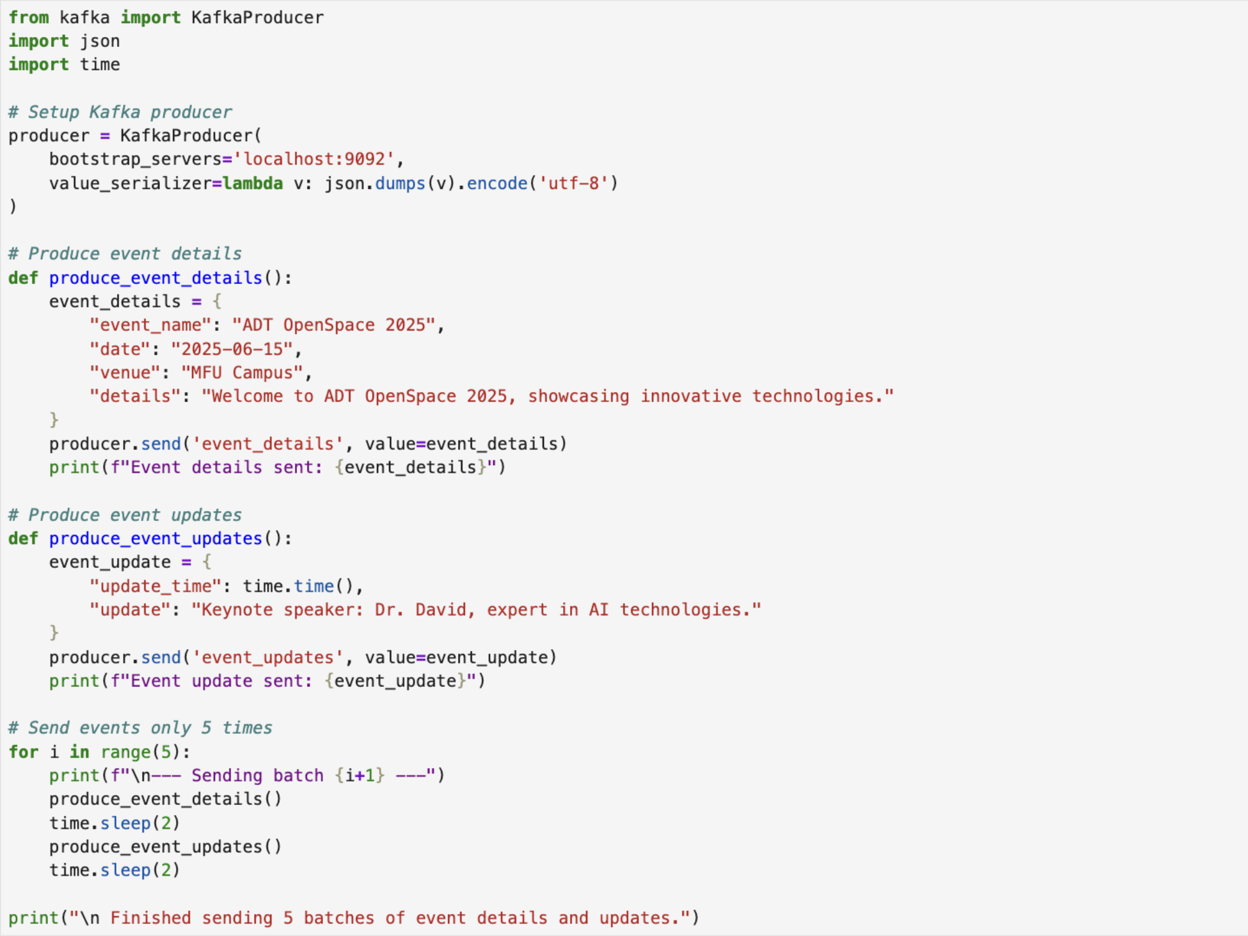
- **Producer:** Sends structured event messages (e.g., event details and updates) to Kafka topics (`event_details` and `event_updates`).
- **Consumer:** Listens to these topics, receives the messages, and prints them out.

Workflow:

1. KafkaProducer connects to the Kafka server and sends JSON-serialized messages to specific topics.
2. KafkaConsumer subscribes to the same topics and continuously listens for new messages.
3. Messages are deserialized and displayed in a readable format.

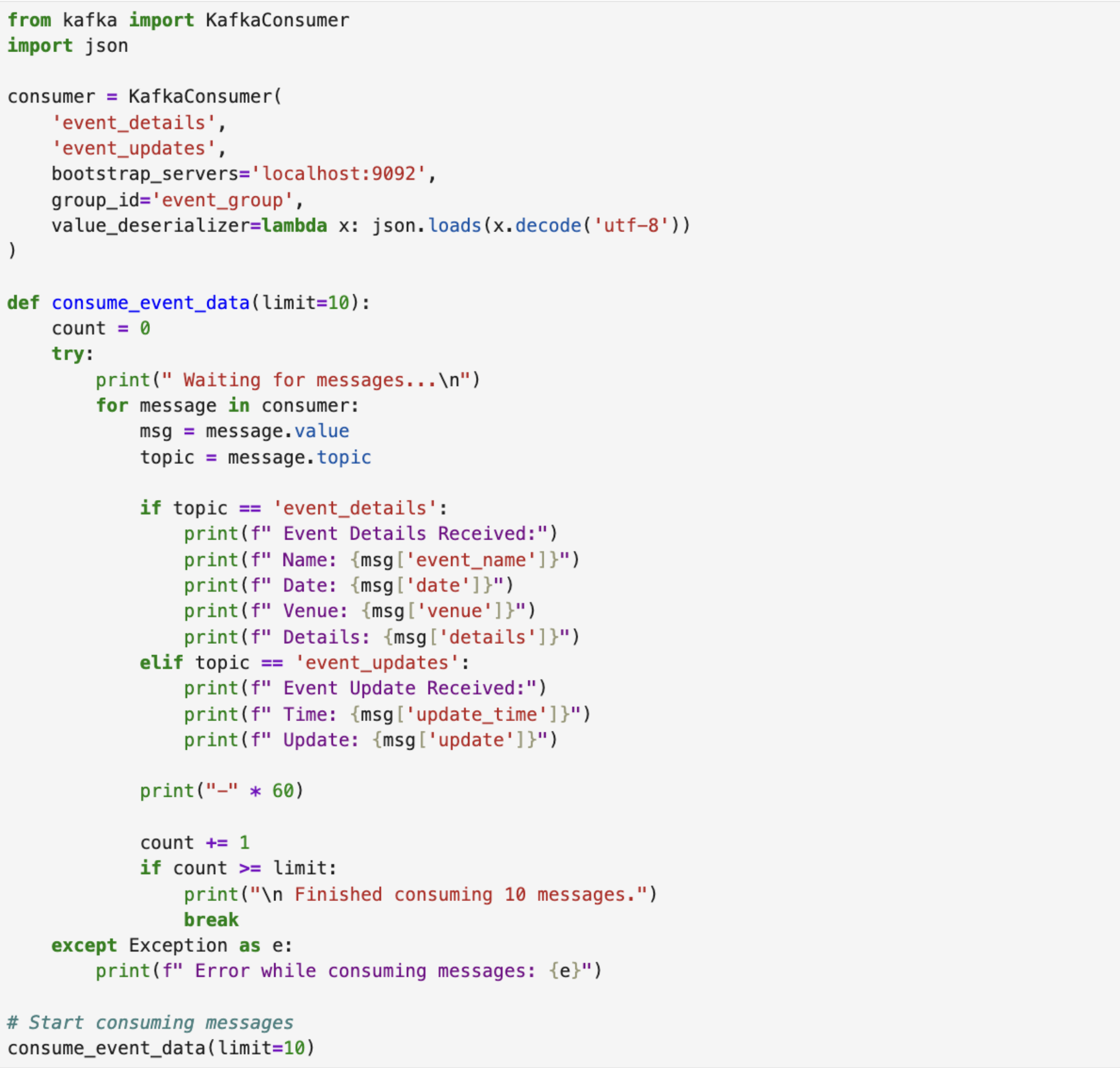
Kafka enables a **decoupled architecture**, where producers and consumers work independently, improving system scalability and reliability.

- **Complete Code /Program**
    - **Producer.ipynb**



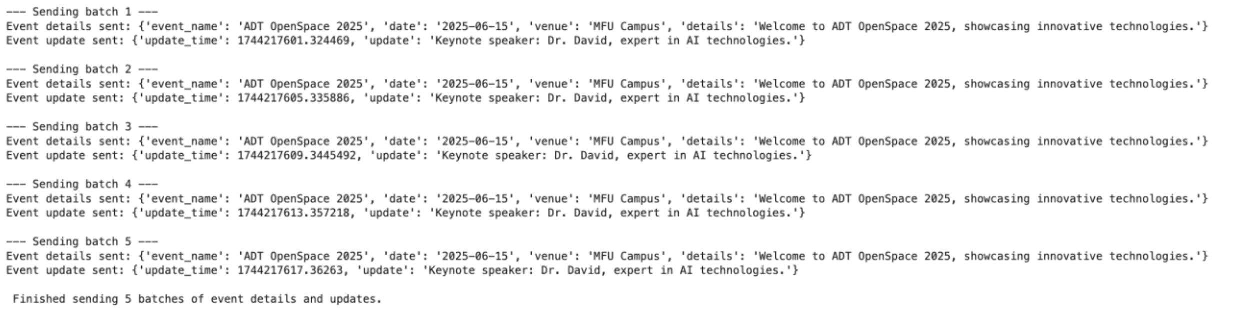
```
from kafka import KafkaProducer
import json
import time

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

# Produce event details
def produce_event_details():
    event_details = {
        "event_name": "ADT OpenSpace 2025",
        "date": "2025-06-15",
        "venue": "MFU Campus",
        "details": "Welcome to ADT OpenSpace 2025, showcasing innovative technologies."
    }
    producer.send('event_details', value=event_details)
    print(f"Event details sent: {event_details}")

# Produce event updates
def produce_event_updates():
    event_update = {
        "update_time": time.time(),
        "update": "Keynote speaker: Dr. David, expert in AI technologies."
    }
    producer.send('event_updates', value=event_update)
    print(f"Event update sent: {event_update}")

# Send events only 5 times
for i in range(5):
    print(f"\n--- Sending batch {i+1} ---")
    produce_event_details()
    time.sleep(2)
    produce_event_updates()
    time.sleep(2)

print("\n Finished sending 5 batches of event details and updates.")
```

- **Consumer.ipynb**



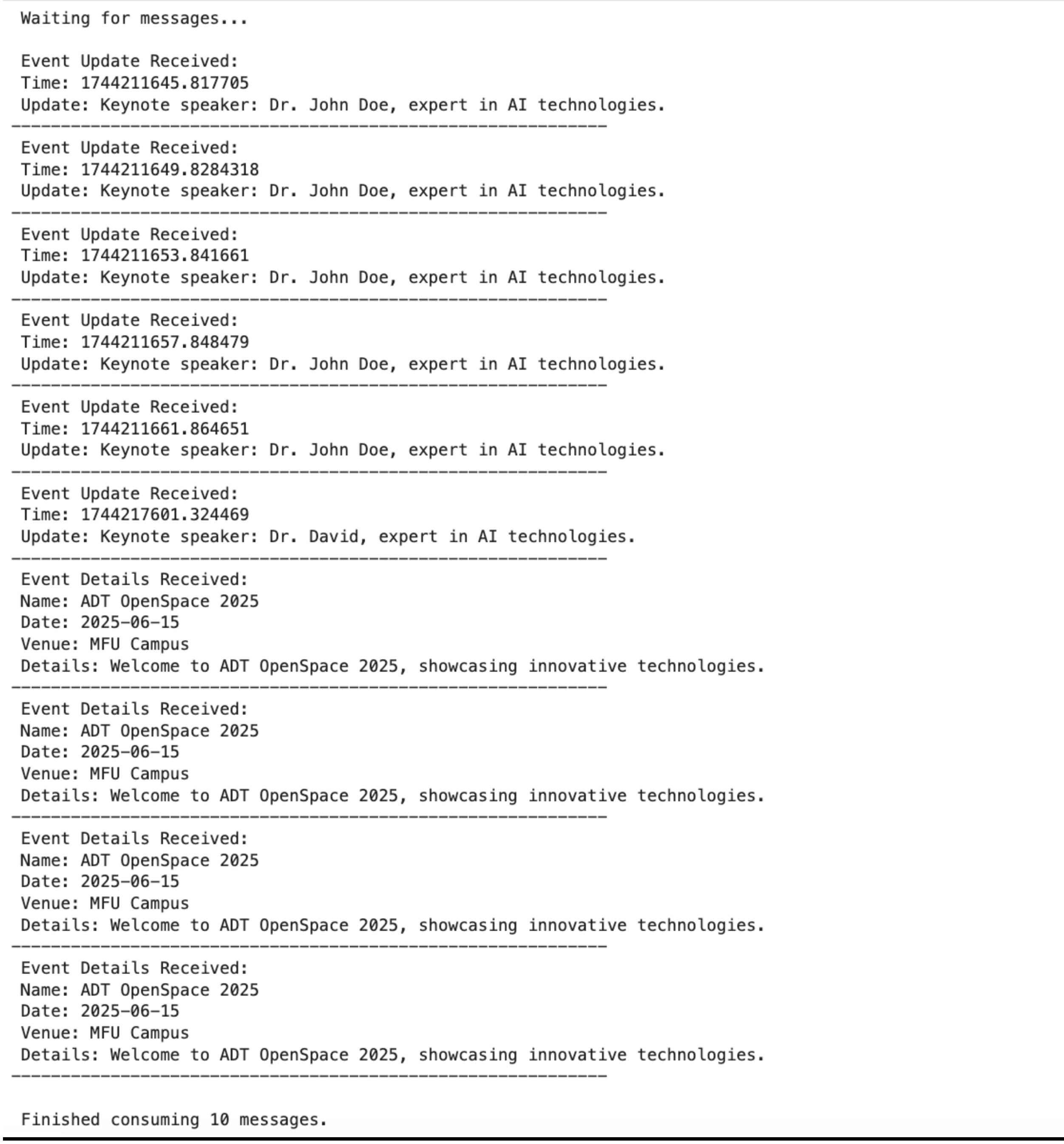
```python
from kafka import KafkaConsumer
import json

consumer = KafkaConsumer(
    'event_details',
    'event_updates',
    bootstrap_servers='localhost:9092',
    group_id='event_group',
    value_deserializer=lambda x: json.loads(x.decode('utf-8'))
)

def consume_event_data(limit=10):
    count = 0
    try:
        print(" Waiting for messages...\n")
        for message in consumer:
            msg = message.value
            topic = message.topic

            if topic == 'event_details':
                print(f" Event Details Received:")
                print(f" Name: {msg['event_name']}")
                print(f" Date: {msg['date']}")
                print(f" Venue: {msg['venue']}")
                print(f" Details: {msg['details']}")
            elif topic == 'event_updates':
                print(f" Event Update Received:")
                print(f" Time: {msg['update_time']}")
                print(f" Update: {msg['update']}")

            print("-" * 60)

            count += 1
            if count >= limit:
                print("\n Finished consuming 10 messages.")
                break
    except Exception as e:
        print(f" Error while consuming messages: {e}")

# Start consuming messages
consume_event_data(limit=10)
```

- **Result**
  - <u>**Producer.ipynb's result**</u>



```
--- Sending batch 1 ---
Event details sent: {'event_name': 'ADT OpenSpace 2025', 'date': '2025-06-15', 'venue': 'MFU Campus', 'details': 'Welcome to ADT OpenSpace 2025, showcasing innovative technologies.'}
Event update sent: {'update_time': 1744217601.324469, 'update': 'Keynote speaker: Dr. David, expert in AI technologies.'}

--- Sending batch 2 ---
Event details sent: {'event_name': 'ADT OpenSpace 2025', 'date': '2025-06-15', 'venue': 'MFU Campus', 'details': 'Welcome to ADT OpenSpace 2025, showcasing innovative technologies.'}
Event update sent: {'update_time': 1744217605.335886, 'update': 'Keynote speaker: Dr. David, expert in AI technologies.'}

--- Sending batch 3 ---
Event details sent: {'event_name': 'ADT OpenSpace 2025', 'date': '2025-06-15', 'venue': 'MFU Campus', 'details': 'Welcome to ADT OpenSpace 2025, showcasing innovative technologies.'}
Event update sent: {'update_time': 1744217609.3445492, 'update': 'Keynote speaker: Dr. David, expert in AI technologies.'}

--- Sending batch 4 ---
Event details sent: {'event_name': 'ADT OpenSpace 2025', 'date': '2025-06-15', 'venue': 'MFU Campus', 'details': 'Welcome to ADT OpenSpace 2025, showcasing innovative technologies.'}
Event update sent: {'update_time': 1744217613.357218, 'update': 'Keynote speaker: Dr. David, expert in AI technologies.'}

--- Sending batch 5 ---
Event details sent: {'event_name': 'ADT OpenSpace 2025', 'date': '2025-06-15', 'venue': 'MFU Campus', 'details': 'Welcome to ADT OpenSpace 2025, showcasing innovative technologies.'}
Event update sent: {'update_time': 1744217617.36263, 'update': 'Keynote speaker: Dr. David, expert in AI technologies.'}

 Finished sending 5 batches of event details and updates.
```

  - <u>**Consumer.ipynb's result**</u>



```
 Waiting for messages...

 Event Update Received:
 Time: 1744211645.817705
 Update: Keynote speaker: Dr. John Doe, expert in AI technologies.
------------------------------------------------------------
 Event Update Received:
 Time: 1744211649.8284318
 Update: Keynote speaker: Dr. John Doe, expert in AI technologies.
------------------------------------------------------------
 Event Update Received:
 Time: 1744211653.841661
 Update: Keynote speaker: Dr. John Doe, expert in AI technologies.
------------------------------------------------------------
 Event Update Received:
 Time: 1744211657.848479
 Update: Keynote speaker: Dr. John Doe, expert in AI technologies.
------------------------------------------------------------
 Event Update Received:
 Time: 1744211661.864651
 Update: Keynote speaker: Dr. John Doe, expert in AI technologies.
------------------------------------------------------------
 Event Update Received:
 Time: 1744217601.324469
 Update: Keynote speaker: Dr. David, expert in AI technologies.
------------------------------------------------------------
 Event Details Received:
 Name: ADT OpenSpace 2025
 Date: 2025-06-15
 Venue: MFU Campus
 Details: Welcome to ADT OpenSpace 2025, showcasing innovative technologies.
------------------------------------------------------------
 Event Details Received:
 Name: ADT OpenSpace 2025
 Date: 2025-06-15
 Venue: MFU Campus
 Details: Welcome to ADT OpenSpace 2025, showcasing innovative technologies.
------------------------------------------------------------
 Event Details Received:
 Name: ADT OpenSpace 2025
 Date: 2025-06-15
 Venue: MFU Campus
 Details: Welcome to ADT OpenSpace 2025, showcasing innovative technologies.
------------------------------------------------------------
 Event Details Received:
 Name: ADT OpenSpace 2025
 Date: 2025-06-15
 Venue: MFU Campus
 Details: Welcome to ADT OpenSpace 2025, showcasing innovative technologies.
------------------------------------------------------------

 Finished consuming 10 messages.
```

- ## Conclusion

This project successfully demonstrates the use of Apache Kafka to handle real-time event streaming. By using **KafkaProducer** and **KafkaConsumer**, the system:

- Efficiently sends and receives structured event data.
- Maintains a scalable and decoupled architecture.
- Can be extended to handle more complex use cases (e.g., multiple producers/consumers, persistent storage, analytics).

This solution is ideal for modern applications that require **real-time communication**, especially in large-scale event management, IoT, and monitoring systems.