Assignment 2B: Using Real-time Streaming Data and Machine Learning to Predict and Visualise eCommerce Fraud

- In assignment 2A, we have already developed the machine learning models
- In part 2B: integrate machine learning model, Kafka and Spark streaming to predict potential fraud in real time

Datasets

All files from assignment 2 Part A(A2A).
Your saved best model from A2A
new_browsing_behaviour.csv: (New browsing behaviour data to simulate real-time streaming)
new_transaction.csv: (New transactions linked to browsing behaviour)

session_id	event_type	event_time	traffic_source	device_type	customer_id
8a4ea821-02ea-4275-8	CL	00:00.5	MOBILE	Android	61923
e224a60c-f02c-4e12-85	CL	00:09.6	MOBILE	Android	52328
8083bc82-d495-4635-b	VI	00:10.6	MOBILE	iOS	33854
3d4fc2ed-23b3-41b9-9	CL	00:18.3	MOBILE	Android	1030
29cab3eb-17b3-472a-8	VP	00:25.7	WEB	Android	12525
7aaab404-e148-484f-b	HP	00:28.8	MOBILE	Android	53754
75666d59-bfd7-42b4-8	SCR	00:29.3	MOBILE	Android	52484
345c167b-79ec-4424-8	ATC	00:29.5	MOBILE	Android	74474
bb769bf5-c21e-450e-a	CL	00:33.5	MOBILE	Android	62950
7b38cf0a-1867-49e4-b	AP	01:08.2	MOBILE	iOS	88618
5c903425-e405-4a70-8	AP	01:10.8	MOBILE	Android	3185
28ac3e09-865b-4761-a	SCR	01:16.6	MOBILE	Android	53288
f85f2f9f-df4a-4aeb-a0a	CO	01:19.4	MOBILE	Android	98123
80c47600-b046-4c9a-9	VP	01:21.2	MOBILE	Android	66808
de1351b6-f248-4917-9	CL	01:26.6	WEB	Android	14318
83fe5f11-fb81-48cd-bf	CL	01:44.5	WEB	iOS	94434
fe137234-c1c7-4d5a-83	VI	01:45.3	MOBILE	iOS	3000
380800ed-8097-484e-8	ATC	01:46.3	MOBILE	Android	99417

Example data from new_browsing_behaviour.csv (customer ID already included in this stream data)

Example data from new_transaction.csv

created_a	customer_id	transaction_id	session_id	product_metadata	payment_	payment_	promo_ar	promo_co	shipment_	shipment_	shipment_	total_amount	clear_payr
01:19.4	98123	97b193ec-7b52-	f85f2f9f-df4a-4	[{'product_id': 9433,	OVO	Fail	0		0	-6.17197	106.7904	271986	0
03:39.9	64835	6d7eb639-bc72-	d03b2432-403a	[{'product_id': 7328,	Credit Car	Success	0		10000	-8.47864	120.8375	428874	1
04:18.2	63101	548045ed-4bbc-	7b373982-d715	[{'product_id': 4990	Debit Carc	Fail	0		0	-7.004	112.2974	132932	0
04:56.4	84108	7d45484f-0d25-	2290c22f-8fda-4	[{'product_id': 3569	LinkAja	Success	4755	WEEKEND	50000	-8.48634	118.9993	229536	1
09:43.3	52484	91b433dd-d15f-	75666d59-bfd7-	[{'product_id': 5997	Credit Car	Fail	0		10000	-1.59265	110.9151	481266	0
11:20.2	73167	a388ca4a-1aeb-	aee50500-e67d-	[{'product_id': 5167	Credit Car	Success	0		10000	-6.47742	106.3696	103723	1
13:37.1	46866	84cb483c-cf47-4	6b6c8909-487a-	[{'product_id': 8124,	OVO	Success	0		10000	-6.29214	106.8978	1685203	1
14:24.7	978	47d6a6d3-590d	249054c2-1016-	[{'product_id': 2751	Debit Carc	Success	6447	LIBURDON	0	-7.35707	108.3407	256243	1
16:21.8	96085	3ed4b54f-0638-	673b581b-6b35	[{'product_id': 5865	LinkAja	Success	6044	WEEKEND	10000	-7.05584	112.1187	188170	1
17:15.0	15602	8d0e3b77-5bf1-	b948da4b-fec5-	[{'product_id': 2459	Debit Carc	Fail	0		10000	-0.80083	123.262	278487	0
19:36.3	66840	09382612-d207-	a89a6765-88ff-4	[{'product_id': 6138,	LinkAja	Success	0		0	-6.15829	106.8795	65563	1
20:11.2	68449	6bf96b67-b2a9-	a87f6892-8b09-	[{'product_id': 5088	OVO	Fail	0		10000	-6.80791	111.1026	319614	0
20:18.1	9710	9588828e-8f59-	947b5b5f-7aa9-	[{'product_id': 3389	Debit Carc	Success	0		0	-8.06462	110.5497	701493	1
21:45.3	65192	3c501b17-18f4-	7db2467a-c905-	[{'product_id': 3231,	Credit Car	Fail	8107	LIBURDON	10000	-8.04399	112.2058	369512	0
22:57.8	8927	c80012a3-0836-	a20899f7-b53d-	[{'product_id': 4909	Credit Car	Fail	0		10000	-5.21836	105.2694	346087	0
23:31.6	82997	451ce2ae-a933-	b8086f32-d838-	[{'product_id': 1563	Credit Car	Success	0		5000	-8.06414	112.5261	159889	1
23:37.1	45114	39f58158-df43-4	3fe68b85-b86a-	[{'product_id': 5861	Gopay	Fail	0		0	-2.48598	119.3739	1071176	0
28:17.1	70528	96c7c6d1-36c1-	3a737027-89d4-	[{'product_id': 3229	Debit Carc	Success	4945	WEEKEND	10000	-7.50412	110.2558	292966	1

Tasks

The MFC company requests a prototype application to ingest the new browsing behaviour data and predict potential fraud.

- **Task 1: Kafka Producer**: You need to simulate the streaming data production using Kafka
- **Task 2: Streaming Application**: Build a streaming application that ingests the data and integrates the machine learning model (trained in A2A) to predict potential fraud.
- **Task 3: Kafka Consumer**: Implement Kafka consumer to consume data from Task 1

1. In task 1, you will read and publish the data to the Kafka stream.
2. In task 2, you will process the data streams using Spark Structured Streaming and PySpark ML/DataFrame.
3. For task 3, you will read the data from the Kafka stream from part 1 and visualise it.

Architecture

The overall architecture of the assignment setup is represented by the following figure.

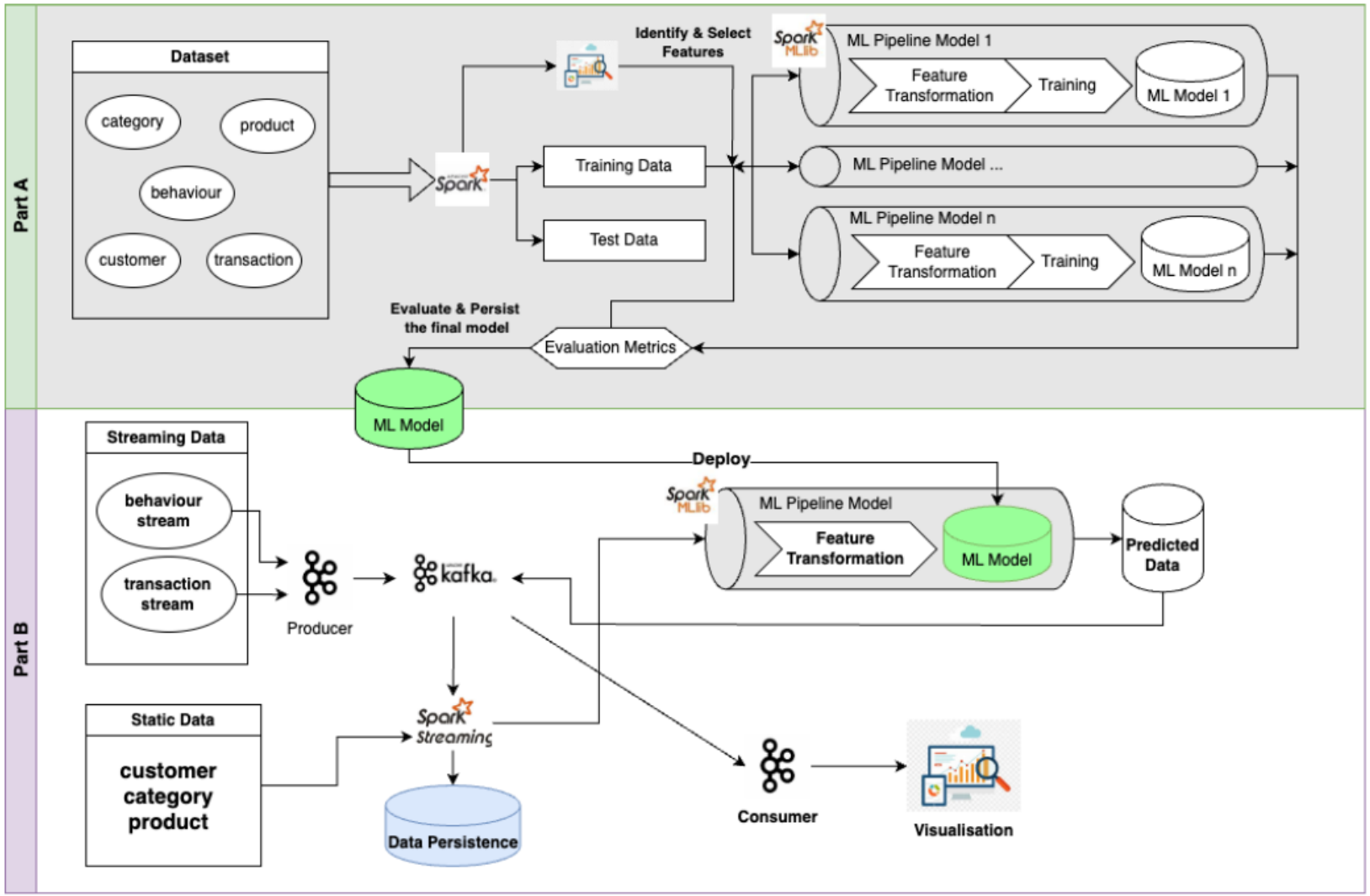


Fig 1: Overall Architecture for Assignment 2 (This assignment is Part B)

Getting started

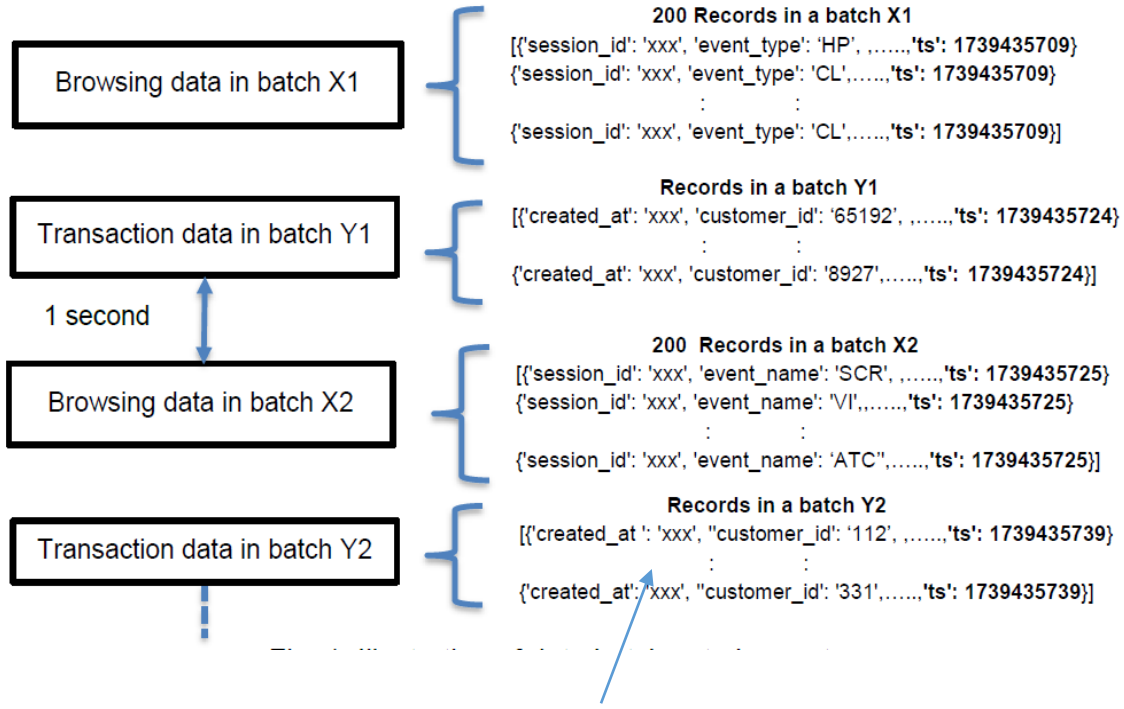
- Download the dataset from Moodle.
- Download three ipynb template files from Moodle

    - ***A2B-Task1_producer.ipynb*** file for streaming data production
    - ***A2B-Task2_spark_streaming.ipynb*** file for consuming and processing data using Spark Structured Streaming
    - ***A2B-Task3_consumer.ipynb*** file for consuming the data using Kafka and visualise

Task 1: Producing the data (15%)

Implement one Kafka producer to simulate real-time data streaming

- Your program should send browsing behaviour data batch-by-batch to the Kafka stream.
  - ✓ One batch consists of a 200 rows from the browsing behaviour dataset.
  - ✓ Keep track of start and end event_time of browsing behaviour data batch (Assume the dataset is sorted by event_time.)
  - ✓ For each row, add a timestamp column named *ts* in Unix timestamp format. This is the event time when you send the streaming data, which should be in *int* format. The *ts* should be the same for the data sent in one batch.
- Read the transaction data rows that fall between the start and end event_time of the browsing behaviour data **(Hint: use 'created_at' time)** recorded in task 1.1, create a batch.
  - ✓ Add the column named 'ts' for each row.
- At an interval of 1 second, send the two data batches consecutively (browsing behaviour & transaction data) to respective Kafka topics



Concatenate records in each batch in the forms of a list of dictionaries

- ✓ All the data except for the ‘ts’ column should be sent in the original string format type, without changing to any other types.

```
from time import sleep
from json import dumps
from kafka3 import KafkaProducer
import csv
import os
from datetime import datetime
```

Example: Part of first-batch data being published


```
[{'session_id': '523091cd-bb1c-4dd8-b7c2-357fd5c73c0b', 'event_type': 'HP', 'event_time': '2024-01-01 00:20:54.302', 'traffic_s
ource': 'MOBILE', 'device_type': 'Android', 'customer_id': '68053', 'ts': 1745818012}, {'session_id': 'b2a2b5f0-554f-4716-b69b-
4e9e28cc5ce2', 'event_type': 'CL', 'event_time': '2024-01-01 00:20:56.922', 'traffic_source': 'MOBILE', 'device_type': 'Androi
d', 'customer_id': '89887', 'ts': 1745818012}, {'session_id': '9357b90c-225f-49f0-9913-ad0be53cf873', 'event_type': 'HP', 'even
t_time': '2024-01-01 00:21:02.377', 'traffic_source': 'MOBILE', 'device_type': 'iOS', 'customer_id': '58931', 'ts': 174581801
2}, {'session_id': 'd798d71e-b01d-4563-a4b2-cd1d98fde4df', 'event_type': 'SCR', 'event_time': '2024-01-01 00:21:10.048', 'traff
ic_source': 'WEB', 'device_type': 'iOS', 'customer_id': '21796', 'ts': 1745818012}, {'session_id': 'd05306a7-237f-4178-8d1d-897
16d864333', 'event_type': 'VI', 'event_time': '2024-01-01 00:21:11.881', 'traffic_source': 'MOBILE', 'device_type': 'Android',
'customer_id': '92509', 'ts': 1745818012}, {'session_id': 'd58640a2-b034-4a4c-9816-3e6bc01c30dd', 'event_type': 'HP', 'event_ti
me': '2024-01-01 00:21:34.208', 'traffic_source': 'MOBILE', 'device_type': 'iOS', 'customer_id': '1698', 'ts': 1745818012}, {'s
ession_id': '7db2467a-c905-4bcb-b2f2-e3a73cec535b', 'event_type': 'CO', 'event_time': '2024-01-01 00:21:45.282', 'traffic_sourc
e': 'MOBILE', 'device_type': 'Android', 'customer_id': '65192', 'ts': 1745818012}, {'session_id': '7fb8e116-6405-4080-b147-33ef
15c44abb', 'event_type': 'HP', 'event_time': '2024-01-01 00:21:47.724', 'traffic_source': 'MOBILE', 'device_type': 'Android',
'customer_id': '20103', 'ts': 1745818012}, {'session_id': '1bf0ec5b-87bb-49df-a5bf-e898f865d6f9', 'event_type': 'CL', 'event_ti
me': '2024-01-01 00:21:54.014', 'traffic_source': 'MOBILE', 'device_type': 'Android', 'customer_id': '48894', 'ts': 174581801
2}, {'session_id': '6c770f51-2cb5-4197-8e5d-e6aaa65f1902', 'event_type': 'CL', 'event_time': '2024-01-01 00:21:56.581', 'traffi
c_source': 'MOBILE', 'device_type': 'Android', 'customer_id': '66067', 'ts': 1745818012}, {'session_id': 'e53cce32-1353-4a64-ab
da-29d03cbe4175', 'event_type': 'CL', 'event_time': '2024-01-01 00:21:59.996', 'traffic_source': 'WEB', 'device_type': 'Androi
```


200 rows of browsing behaviour data


```
[{'created_at': '2024-01-01 00:21:45.282', 'customer_id': '65192', 'transaction_id': '3c501b17-18f4-485b-9a98-9bbe2a8309d8', 's
ession_id': '7db2467a-c905-4bcb-b2f2-e3a73cec535b', 'product_metadata': "[{'product_id': 3231, 'quantity': 1, 'item_price': 367
619}]", 'payment_method': 'Credit Card', 'payment_status': 'Fail', 'promo_amount': '8107', 'promo_code': 'LIBURDONG', 'shipment
_fee': '10000', 'shipment_location_lat': '-8.04398660050412', 'shipment_location_long': '112.205799773991', 'total_amount': '36
9512', 'clear_payment': '0', 'ts': 1745818018}, {'created_at': '2024-01-01 00:22:57.790', 'customer_id': '8927', 'transaction_i
d': 'c80012a3-0836-4b35-9039-84b400c850eb', 'session_id': 'a20899f7-b53d-4513-a220-a625c9d5315d', 'product_metadata': "[{'produ
ct_id': 49090, 'quantity': 1, 'item_price': 336087}]", 'payment_method': 'Credit Card', 'payment_status': 'Fail', 'promo_amoun
t': '0', 'promo_code': '', 'shipment_fee': '10000', 'shipment_location_lat': '-5.21835854597248', 'shipment_location_long': '10
5.269397530302', 'total_amount': '346087', 'clear_payment': '0', 'ts': 1745818018}, {'created_at': '2024-01-01 00:23:31.605',
'customer_id': '82997', 'transaction_id': '451ce2ae-a933-4635-9fa4-8ab47586b7fa', 'session_id': 'b8086f32-d838-4c65-81d6-3fa44d
7e6545', 'product_metadata': "[{'product_id': 15631, 'quantity': 1, 'item_price': 154889}]", 'payment_method': 'Credit Card',
'payment_status': 'Success', 'promo_amount': '0', 'promo_code': '', 'shipment_fee': '5000', 'shipment_location_lat': '-8.064143
```


Varied number of rows of transaction data (depending on how many rows fall start and end event_time of the browsing behaviour data)

Task 2: Streaming application using Spark Structured Streaming (70%)

Implement Spark Structured Streaming to consume the data from task 1 and perform predictive analytics

2.1 Create SparkSession

1. Write code to create a SparkSession using a SparkConf object, which uses **four cores** with **a proper application name**, and use the Melbourne timezone to ensure a checkpoint location has been set.

2.2 Define dataframe loading schema & Load CSV files

2. Similar to assignment 2A, write code to define the data schema for the data files, following the data types suggested in the metadata file. Load the static datasets (e.g. customer, product, category) into data frames. (You can use your code from 2A.)

2.3 Ingesting data

3. Using the Kafka topic from the producer in Task 1, ingest the streaming data into Spark Streaming, assuming all data comes in the **String** format. Except for the 'ts' column, you shall receive it as an **Int** type.

2.4 Transforming data format

4. Then, the streaming data format should be transformed into the proper formats following the metadata file schema, similarly to assignment 2A. Perform the following tasks:
   a) For the 'ts' column, convert it to the timestamp format, we will use it as **event_time**.
   b) If the data is late for more than 1 minute, discard it. (Hint: Use watermaking)

product_metadata

created_a	customer_id	transaction_id	session_id	product_metadata	payment_
01:19.4	98123	97b193ec-7b52-	f85f2f9f-df4a-4	[{'product_id': 9433, 'quantity': 1, 'item_price': 271986}]	OVO
03:39.9	64835	6d7eb639-bc72	d03b2432-403a	[{'product_id': 7328, 'quantity': 2, 'item_price': 209437}]	Credit Car
04:18.2	63101	548045ed-4bbc	7b373982-d715	[{'product_id': 49906, 'quantity': 1, 'item_price': 132932}]	Debit Carc
04:56.4	84108	7d45484f-0d25-	2290c22f-8fda-4	[{'product_id': 35696, 'quantity': 1, 'item_price': 184291}]	LinkAja
09:43.3	52484	91b433dd-d15f	75666d59-bfd7-	[{'product_id': 59970, 'quantity': 1, 'item_price': 471266}]	Credit Car
11:20.2	73167	a388ca4a-1aeb-	aee50500-e67d	[{'product_id': 51671, 'quantity': 1, 'item_price': 93723}]	Credit Car
13:37.1	46866	84cb483c-cf47-4	6b6c8909-487a-	[{'product_id': 8124, 'quantity': 1, 'item_price': 316502}, {'product_id': 18879, 'quantity': 1, 'item_price': 27693	OVO
14:24.7	978	47d6a6d3-590d	249054c2-1016-	[{'product_id': 27514, 'quantity': 1, 'item_price': 262690}]	Debit Carc
16:21.8	96085	3ed4b54f-0638-	673b581b-6b35	[{'product_id': 58653, 'quantity': 1, 'item_price': 184214}]	LinkAja
17:15.0	15602	8d0e3b77-5bf1-	b948da4b-fec5-	[{'product_id': 24596, 'quantity': 1, 'item_price': 268487}]	Debit Carc
19:36.3	66840	09382612-d207	a89a6765-88ff-4	[{'product_id': 6138, 'quantity': 1, 'item_price': 65563}]	LinkAja

2.5 Prepare features & Join local data

- Aggregate the streaming data frame by session id and create the features you used in your assignment 2A model. (note: customer ID has already been included in the stream.)
- Then, join the **static** data frames with the streaming data frames as the final data for prediction.
- Perform data type/column conversion according to your ML model and print out the Schema. (Again, you can reuse code from 2A).

Hints: You can refer spark documentation for examples of **stream-static & stream-stream join**

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#stream-stream-joins

For stream-stream join, you need to apply watermarking to one/both streams based on event time

2.6 Load ML model & Perform Prediction

✓ Load your machine learning model, and use the model to predict if there is a fraud or not in each browsing session/transaction

Hints:

- ❑ Use extractall() method will extract all the contents of your machine learning model zip file
- ❑ The resulting folder “my_ML_model” should contain two sub-folders: “metadata” and “stages”
- ❑ The machine learning model has been trained, you can directly use your trained model to do prediction, using the streaming dataframe as test data (**Notes: the final streaming dataframe here should contains both original data columns and created features, do NOT delete the original data columns since the PipelineModel (combined both Feature transformation + ML) will take them to output prediction**)
- ❑ Use load() method under PipelineModel to load the machine learning model

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.PipelineModel.html#

- ✓ Persist the prediction result in parquet format, then read the parquet result and show the results.

Example of prediction results showed (from lab 5)

rawPrediction	probability	prediction
[0.70265140321958...	[0.66877536002484...	0.0
[0.05994426149296...	[0.51498157951466...	0.0
[3.09851249236433...	[0.95683134510941...	0.0
[2.82983618225742...	[0.94426698148712...	0.0
[2.00240866080951...	[0.88104974004571...	0.0
[0.78471255869873...	[0.68669488839868...	0.0
[3.86580663806994...	[0.97948371475243...	0.0
[0.27189533856394...	[0.56755814851829...	0.0
[-1.5179830427396...	[0.17975871865726...	1.0
[3.79709998033862...	[0.97805657531244...	0.0
[1.24552022146082...	[0.77652342684406...	0.0
[-1.3858554658816...	[0.20007023248440...	1.0
[-0.9015156506619...	[0.28873913030981...	1.0
[0.67089125407154...	[0.66170269716509...	0.0
[-0.4208458985314...	[0.39631435177549...	1.0
[0.60739844738801...	[0.64734712460201...	0.0
[-0.4501270967131...	[0.38933054809342...	1.0
[0.77377960670314...	[0.68433793086694...	0.0
[2.07545846671577...	[0.88849488899444...	0.0
[1.97493940014817...	[0.87814066461078...	0.0

2.7 Analyse prediction results

a) Every 10 seconds, show the total number of frauds (prediction = 1) in the last 2 minutes.

- ❑ Hints: Use window size of 2 minutes with 10-second slide
- ❑ Show results every 10 seconds (Use trigger interval of 10 seconds).

Hint: Use foreachBatch as output sink.

Example Output

```
+------------------------------------------+-----------+
|window                                    |fraud_count|
+------------------------------------------+-----------+
|{2025-02-13 13:51:00, 2025-02-13 13:53:00}|1          |
|{2025-02-13 13:51:10, 2025-02-13 13:53:10}|1          |
+------------------------------------------+-----------+
```

(You can stop this stream after some results were showed)

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#foreachbatch

b) Every 30 seconds, show the total quantity of products for non-fraud transactions (i.e., when prediction=0). You can assess to this information by extracting from the customer shopping cart detail from product metadata in the transaction data.

c) Every 1 minute, find the top 20 products by total quantity. Show their product ID, name and total quantity. Hints: You can also use product_metadata, and for non-fraud transactions only

Task 3: Consuming data using Kafka & Visaulize (15%)

Implement Apache Kafka consumer to consume the data from task 1.

Important:

- **In this part, Kafka consumers are used to consume the streaming data published from Part 1.**
- **Please do not use Spark in this part. It's OK to use Pandas or any Python library to do simple calculations for the visualisation.**

```
from kafka import KafkaConsumer
import matplotlib.pyplot as plt
import datetime as dt
import json
# this line is needed for the
# inline display of graphs in Jupyter Notebook
%matplotlib notebook
```

- ❑ Write a python program that consumes the streaming transaction data sent by the producer in Part 1 using kafka consumer.

- ❑ Your program should also get the number of successful transactions (i.e., when 'payment_status' is 'Success') for each batch of transaction data, and visualise these counts in real time based on the event timestamp.
  a) Use line charts to visualize it.
  b) Please use ts as timestamp generated from producer step.
  c) This plot shall be real-time plot