Question 1:

1) Write an SQL statement to find number of buyer of each country that purchased item with

even and odd itemid number

\*\*notes: itemid data type is integer

Fix the column type first for SQL test

ALTER TABLE orders

ALTER COLUMN shopid TYPE VARCHAR;

ALTER TABLE performance

ALTER COLUMN shopid TYPE VARCHAR;

ALTER TABLE orders

ALTER COLUMN buyerid TYPE VARCHAR;

ALTER TABLE orders

ALTER COLUMN orderid TYPE VARCHAR;

ALTER TABLE orders

ALTER COLUMN itemid TYPE int;

ALTER TABLE quiros.public.user

ALTER COLUMN shopid TYPE VARCHAR;

ALTER TABLE quiros.public.user

ALTER COLUMN buyerid TYPE VARCHAR;

-- question 1

with even as (

select u.country ,count(distinct o.buyerid) "even\_buyer"

from orders o

left join (select distinct buyerid ,country from "user") u on o.buyerid = u.buyerid

where o.itemid % 2 = 0 group by u.country),

odd as (select u.country ,count(distinct o.buyerid) "odd\_buyer"

from orders o

left join (select distinct buyerid ,country from "user") u on o.buyerid = u.buyerid

where o.itemid % 2 = 1 group by u.country )

select distinct u2.country, e.even\_buyer, od.odd\_buyer

from "user" u2

left join even e on u2.country = e.country

left join odd od on u2.country = od.country;

2) Write an SQL statement to find the number of order/views & clicks/impressions of each shop

(if possible make it in 1 query)

-- question 2

with g\_ord as (select o.shopid ,count(o.orderid) "total\_order"

from orders o group by 1),

g\_view as(select p.shopid , sum(p."Item\_views") "total\_view"

from performance p group by 1),

g\_cli\_imp as (select p.shopid , sum(p.total\_clicks) "total\_click",sum(p.impressions) "total\_impression"

from performance p group by 1)

select go.shopid, go.total\_order, vi.total\_view, gci.total\_click, gci.total\_impression

from g\_ord go

left join g\_view vi

on go.shopid = vi.shopid

left join g\_cli\_imp gci

on go.shopid = gci.shopid;

3) Write an SQL statement to find the TOP 10 Buyer by GMV in Country ID & SG

-- question 3

select o.buyerid ,sum(o.gmv)"total\_gmv" from orders o

inner join (select distinct buyerid , country from "user") u on u.buyerid = o.buyerid

where u.country = 'ID'

group by 1

order by 2 desc

limit 10

;

-- question 3

select o.buyerid ,sum(o.gmv)"total\_gmv" from orders o

inner join (select distinct buyerid , country from "user") u on u.buyerid = o.buyerid

where u.country = 'SG'

group by 1

order by 2 desc

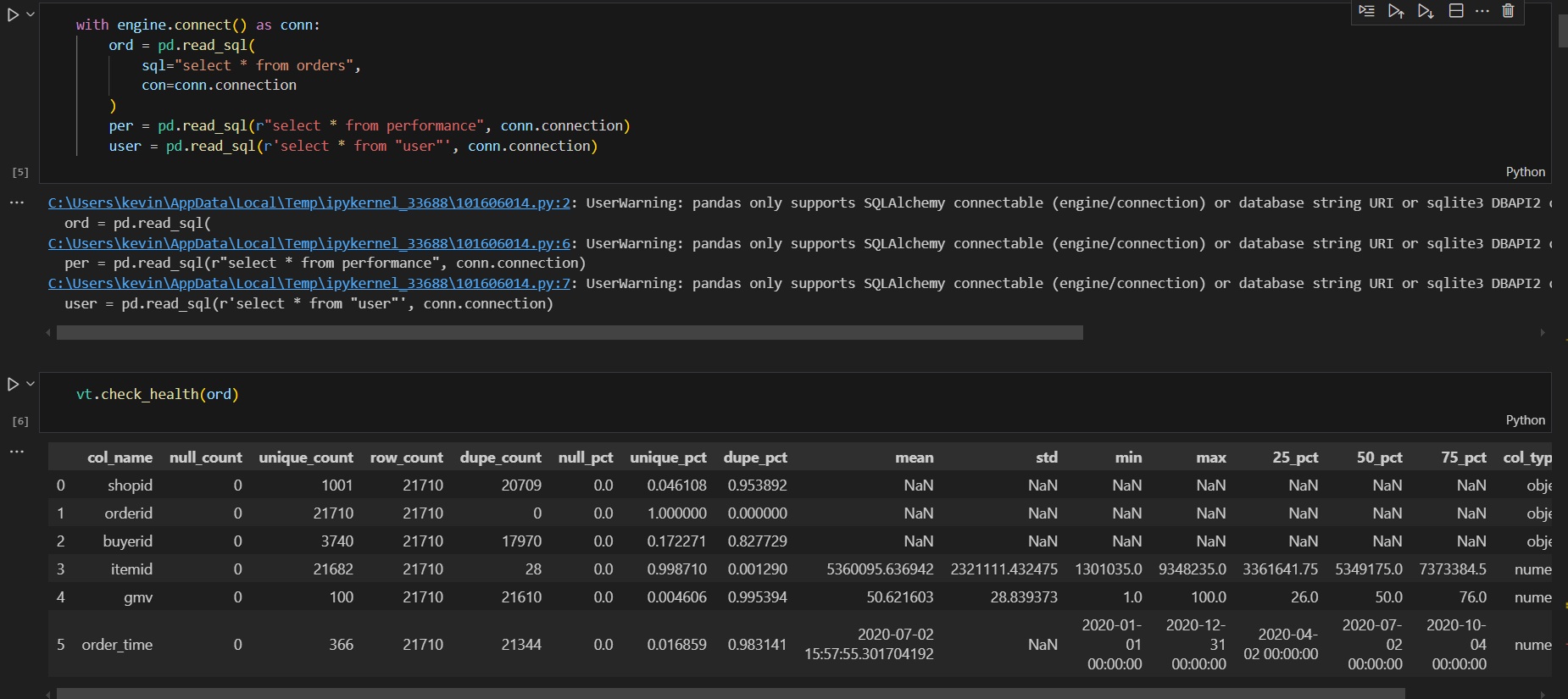
limit 10

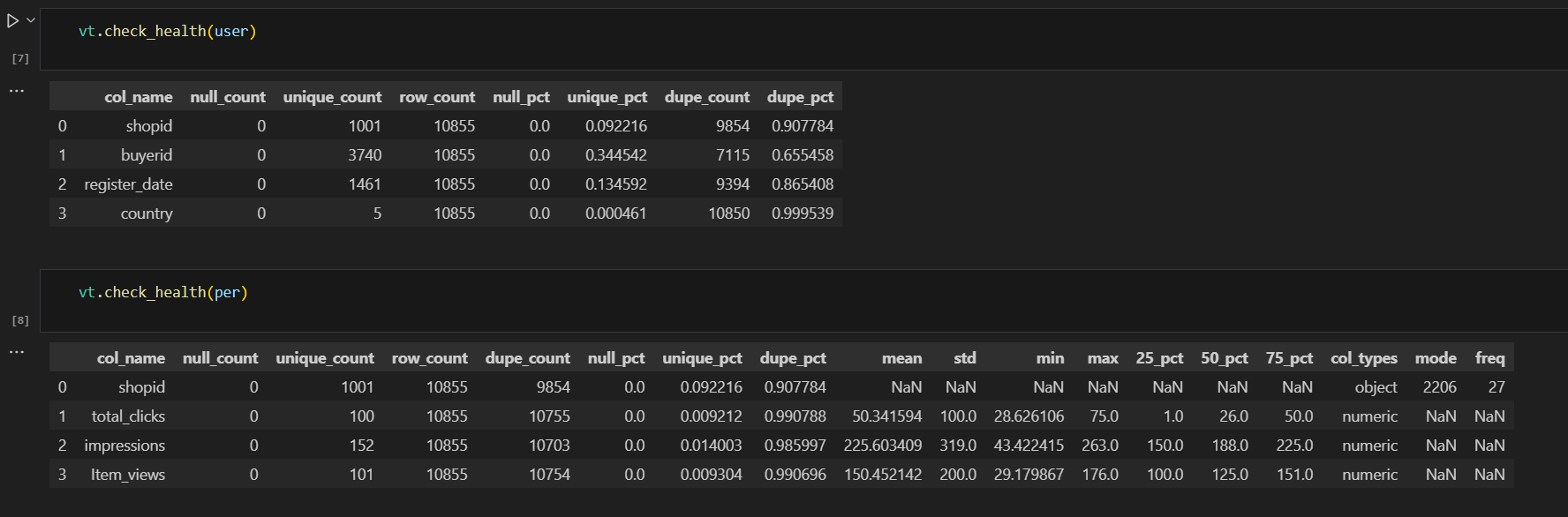
;

Note: you can check the query on **quiros.session.sql** file in the project source code

4) Find out what is wrong with the sample data

My inspection for the data





Summary:

1. There are nothing wrong with the null values, all data is complete in terms of business unique key
2. The unique id for order is unique (no duplicacy)
3. Performance table is also normal
4. **buyer/user table has unnecessary column “shop\_id” hence yielding duplication of user information related**

5) Write a Pandas program to read the given excel data (maven\_music\_customers.csv) and

(maven\_music\_listening\_history.xlsx) into a Pandas dataframe.

mvn\_cust = pd.read\_csv("Files for Technical Test - New\Files for Technical Test\Files for Python Test\maven\_music\_customers.csv")

mvn\_hist = pd.read\_excel("Files for Technical Test - New\Files for Technical Test\Files for Python Test\maven\_music\_listening\_history.xlsx")

6) Clean the data by converting data types, resolving data issues, and creating new columns

7) Explore the data independently, then join the tables for further exploration

For question 6 & 7

def transfrom\_maven():

mvn\_cust = pd.read\_csv("Files for Technical Test - New\Files for Technical Test\Files for Python Test\maven\_music\_customers.csv")

mvn\_hist = pd.read\_excel("Files for Technical Test - New\Files for Technical Test\Files for Python Test\maven\_music\_listening\_history.xlsx")

#customer id to string

mvn\_cust['Customer ID'] = mvn\_cust['Customer ID'].astype(str)

#convert date column to datetime

mvn\_cust['Member Since'] = pd.to\_datetime(mvn\_cust['Member Since'])

mvn\_cust['Cancellation Date'] = pd.to\_datetime(mvn\_cust['Cancellation Date'])

#fill discount with No

mvn\_cust['Discount?'] = mvn\_cust['Discount?'].fillna('No')

#Subscription Plan fill with Basic(Ads)

mvn\_cust['Subscription Plan'] = mvn\_cust['Subscription Plan'].fillna('Basic (Ads)')

#create column is\_active from cancellation date

mvn\_cust['is\_active'] = mvn\_cust['Cancellation Date'].isnull().map({True: 1, False: 0})

#create new column member\_duration, which take the difference between cancellation date and member since and fill with current date

mvn\_cust['member\_duration'] = mvn\_cust['Cancellation Date'] - mvn\_cust['Member Since']

mvn\_cust['member\_duration'] = mvn\_cust['member\_duration'].fillna(datetime.datetime.now() - mvn\_cust['Member Since'])

mvn\_cust['member\_duration'] = mvn\_cust['member\_duration'].dt.days

#convert Customer ID to string

mvn\_hist['Customer ID'] = mvn\_hist['Customer ID'].astype(str)

#convert Session ID to string

mvn\_hist['Session ID'] = mvn\_hist['Session ID'].astype(str)

#convert Audio ID to string

mvn\_hist['Audio ID'] = mvn\_hist['Audio ID'].astype(str)

#get total session per customer

mvn\_hist\_grouped = mvn\_hist.groupby('Customer ID').agg({'Session ID': 'nunique'}).reset\_index()

mvn\_hist\_grouped.columns = ['Customer ID', 'Total Session']

#get total song played per customer

mvn\_hist\_grouped\_audio = mvn\_hist.groupby('Customer ID').agg({'Audio ID': 'count'}).reset\_index()

mvn\_hist\_grouped\_audio.columns = ['Customer ID', 'Total Audio']

#get average song played per session per customer

mvn\_hist\_grouped\_audio\_per\_session = mvn\_hist.groupby('Customer ID').agg({'Audio ID': 'count', 'Session ID': 'nunique'}).reset\_index()

mvn\_hist\_grouped\_audio\_per\_session['Average Audio per Session'] = mvn\_hist\_grouped\_audio\_per\_session['Audio ID'] / mvn\_hist\_grouped\_audio\_per\_session['Session ID']

mvn\_hist\_grouped\_audio\_per\_session = mvn\_hist\_grouped\_audio\_per\_session[['Customer ID', 'Average Audio per Session']]

#get total Podcast and song in audio type per customer id

mvn\_hist\_grouped\_audio\_type = mvn\_hist.groupby(['Customer ID', 'Audio Type']).size().reset\_index(name='Count')

mvn\_hist\_grouped\_audio\_type = mvn\_hist\_grouped\_audio\_type.pivot(index='Customer ID', columns='Audio Type', values='Count').reset\_index()

mvn\_hist\_grouped\_audio\_type = mvn\_hist\_grouped\_audio\_type.fillna(0)

mvn\_hist\_grouped\_audio\_type.columns = ['Customer ID', 'total\_Podcast', 'total\_Song']

#get average Podcast and song in audio type per session per customer id

mvn\_hist\_grouped\_audio\_type\_per\_session = mvn\_hist.groupby(['Customer ID', 'Audio Type']).agg({'Audio ID': 'count', 'Session ID': 'nunique'}).reset\_index()

mvn\_hist\_grouped\_audio\_type\_per\_session['Average Audio per Session'] = mvn\_hist\_grouped\_audio\_type\_per\_session['Audio ID'] / mvn\_hist\_grouped\_audio\_type\_per\_session['Session ID']

mvn\_hist\_grouped\_audio\_type\_per\_session = mvn\_hist\_grouped\_audio\_type\_per\_session.pivot(index='Customer ID', columns='Audio Type', values='Average Audio per Session').reset\_index()

mvn\_hist\_grouped\_audio\_type\_per\_session = mvn\_hist\_grouped\_audio\_type\_per\_session.fillna(0)

#rename column

mvn\_hist\_grouped\_audio\_type\_per\_session.columns = ['Customer ID', 'avg\_Podcast\_per\_session', 'avg\_Song\_per\_session']

#merge all dataframe

output = mvn\_cust.merge(mvn\_hist\_grouped, on='Customer ID', how='left')\

.merge(mvn\_hist\_grouped\_audio, on='Customer ID', how='left')\

.merge(mvn\_hist\_grouped\_audio\_per\_session, on='Customer ID', how='left')\

.merge(mvn\_hist\_grouped\_audio\_type, on='Customer ID', how='left')\

.merge(mvn\_hist\_grouped\_audio\_type\_per\_session, on='Customer ID', how='left')

output.to\_parquet(r'output\raw\maven\_raw.parquet',index=False,engine='fastparquet')

8) Create a non-null, numeric DataFrame and engineer features that could be good predictors of

customer churn

def clean\_maven():

raw\_maven = pd.read\_parquet(r"output\raw\maven\_raw.parquet")

#map subscription plan to 0 and 1 with 0 is Basic (Ads) and 1 is Premium

raw\_maven['Subscription Plan'] = raw\_maven['Subscription Plan'].map({'Basic (Ads)': 0, 'Premium (No Ads)': 1})

#convert subscription rate to float

raw\_maven['Subscription Rate'] = raw\_maven['Subscription Rate'].str.replace('$', '').astype(float)

#map discount to 0 and 1 with 0 is No and 1 is Yes

raw\_maven['Discount?'] = raw\_maven['Discount?'].map({'No': 0, 'Yes': 1})

# Get only numeric features from raw\_maven

clean = raw\_maven[['Subscription Plan',

'Subscription Rate',

'is\_active',

'member\_duration',

'Total Session',

'Total Audio',

'Average Audio per Session',

'total\_Podcast',

'total\_Song',

'avg\_Podcast\_per\_session',

'avg\_Song\_per\_session']]

clean.to\_parquet(r'output\cleaned\maven\_clean.parquet',index=False,engine='fastparquet')

Case Scenario DWH

1. Create a star schema to organize the data efficiently

Please check the Quiros Supporting-ERD DWH.jpg or open the .drawio diagrams in the project deliverables or open <https://drive.google.com/drive/folders/1UgbOlYk_8CXpfjZhsZJE261myqMIhwKH?usp=sharing>

1. Create data pipeline to handle both batching and streaming

Pipeline for batch processing

def batch\_cdr():

#case1: updated data

#case2: added new data

#case3: deleted data

#get cdr\_modified\_date from last 3 days

existing\_clean\_cdr\_path = r"batch\_job\cdr.parquet"

new\_batch\_cdr\_path = r"dwh\cdr\cdr\_case1.parquet"

output\_clean\_cdr\_path = r"batch\_job\cdr.parquet"

df\_before = pd.read\_parquet(existing\_clean\_cdr\_path)

df\_after = pd.read\_parquet(new\_batch\_cdr\_path)

df\_before['cdr\_modified\_date'] = pd.to\_datetime(df\_before['cdr\_modified\_date'])

df\_after['cdr\_modified\_date'] = pd.to\_datetime(df\_after['cdr\_modified\_date'])

#get last 3 days modified data

updated = df\_after[df\_after['cdr\_modified\_date'] > datetime.datetime.now() - datetime.timedelta(days=3)]

#check if updated has data or not

if updated.shape[0] > 0:

#unique business id

u\_cdr = df\_after['cdr\_id'].unique()

to\_update = df\_after[df\_after['cdr\_id'].isin(updated['cdr\_id'])]

#update old data with updated data, covers updated data and new data

df\_before = df\_before[~df\_before['cdr\_id'].isin(updated['cdr\_id'])]

df\_before = pd.concat([df\_before, to\_update])

#inner join df\_before u\_cdr to cover deleted data

df\_before = df\_before[df\_before['cdr\_id'].isin(u\_cdr)]

df\_before.to\_parquet(output\_clean\_cdr\_path, index=False)

return "data is updated"

return "data is updated"

Note: you can view the source code on main\_batch.py

Pipeline for streaming

from pyspark.sql import SparkSession

from pyspark.sql.functions import col

#pyspark import unix\_timestamp

from pyspark.sql.functions import unix\_timestamp

#pyspark import TimestampType

from pyspark.sql.types import TimestampType

import pandas as pd

# Step 1: Set up the Spark session

spark = SparkSession.builder \

.appName("Streaming JSON Processing") \

.getOrCreate()

spark.sql("set spark.sql.streaming.schemaInference=true")

# Step 2: Define the streaming source (e.g., a directory)

input\_directory = r"dwh\transaction"

# Step 3: Read the streaming JSON data into a PySpark DataFrame

streaming\_df = spark.readStream \

.format("json") \

.option("path", input\_directory) \

.option("maxFilesPerTrigger", 1) \

.load()

processed\_df = streaming\_df.select("customer\_id", "transaction\_price")

#write datafram to csv file

processed\_df.writeStream \

.outputMode("append") \

.format("json") \

.option("path", r"dwh\transaction\stream\_out.json") \

.option("checkpointLocation", "checkpoint") \

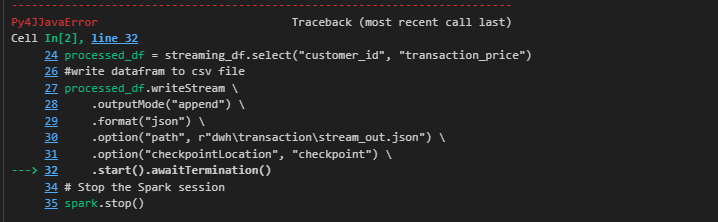
.start().awaitTermination()

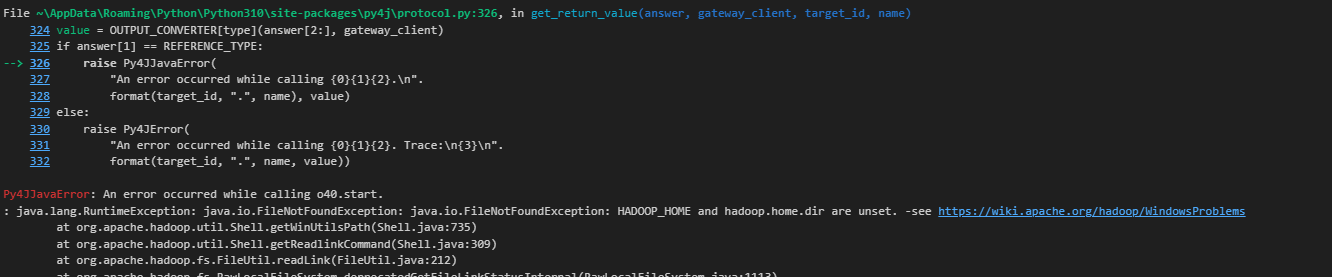
# Stop the Spark session

spark.stop()

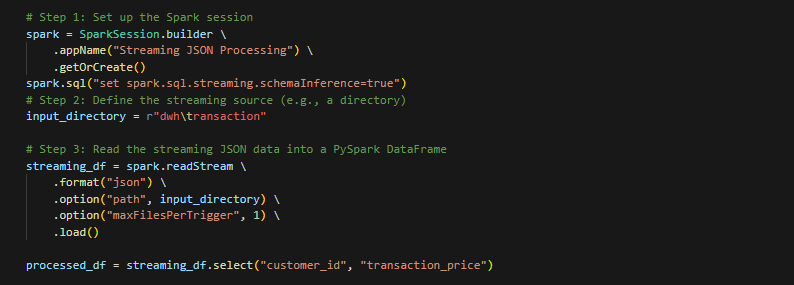
Unfortunately, the streaming pipeline cannot be run because:

* Failed Windows OS deployment
* Failed tools dependency (spark,hadoop, winutils.exe)



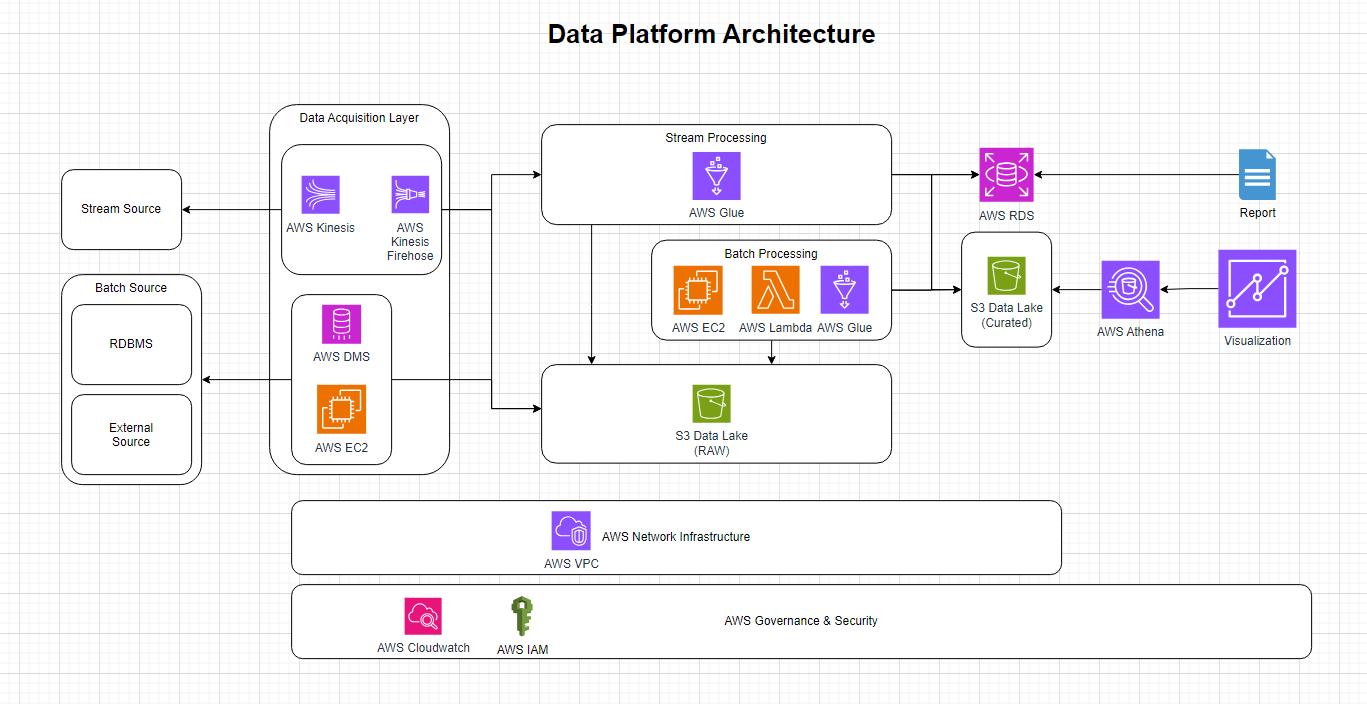


But the read and process pipeline is a success, the pipeline only failed while exporting the transformation process



Note: you can see the source code on main\_stream.py in project deliverables

1. Create strategies for optimization techniques
   1. Micro-batching and more frequent update pipeline refresh.=
   2. Only store and define “id” related column as integer due to efficient memory size
   3. Store data as parquet format
   4. In terms of cost optimization, choose the right tools for the right case. Example: use batch tooling set for only batch case purpose not for handling streaming purpose, and vice versa
   5. Use distributed data processing tools to process the data in distributed computing hence make it faster. Tools like apache spark/pyspark
   6. Use distributed computing infrastructure to handle the computation workload distributedly across multiple hardware. Solution like apache yarn on top of multiple server/AWS EC2
   7. Design high availability, durable, and scalable infrastructure



Project github: <https://github.com/zylbergs/telmark>

Gdrive: <https://drive.google.com/drive/folders/1UgbOlYk_8CXpfjZhsZJE261myqMIhwKH>