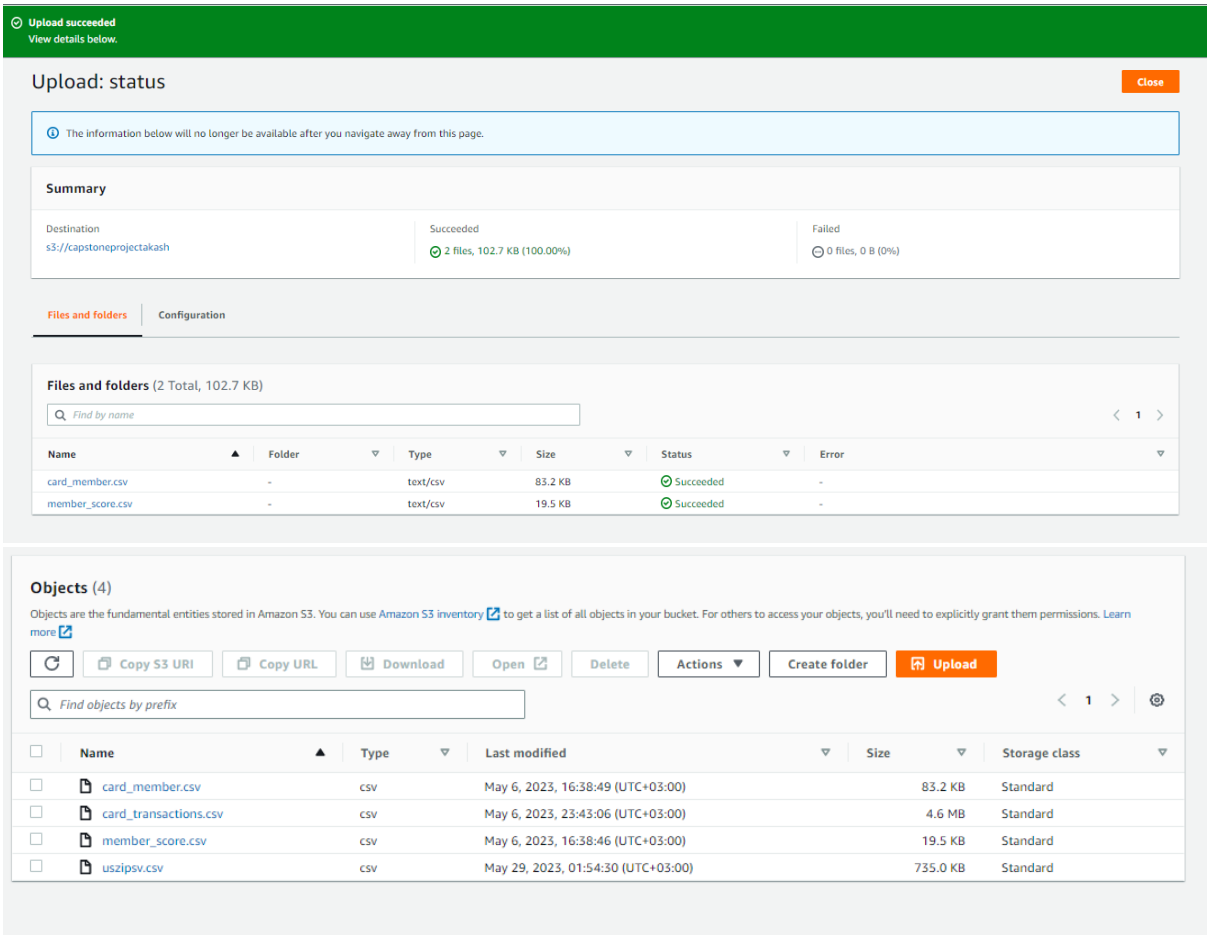
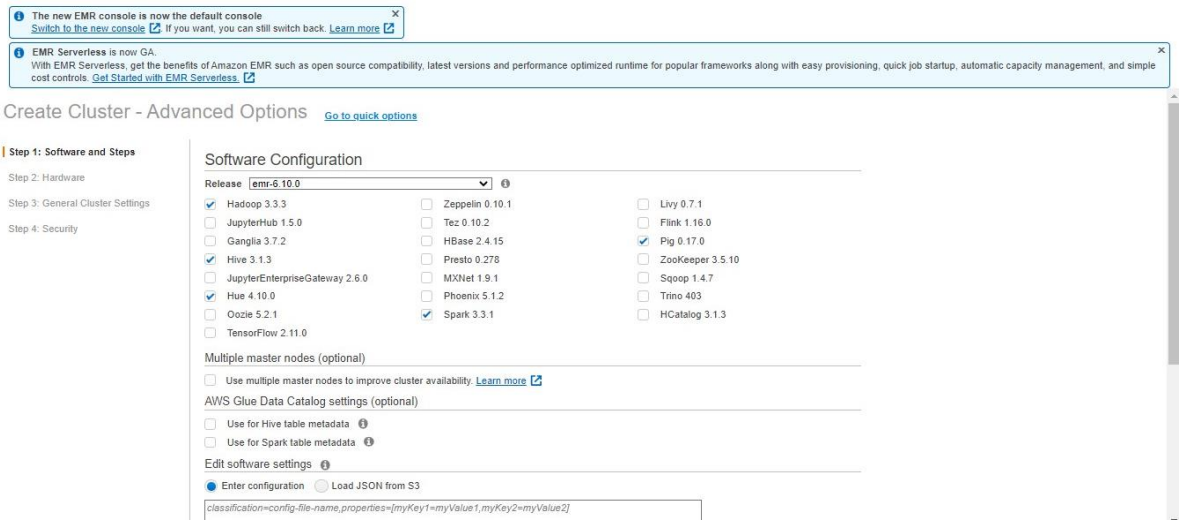
Credit Card Fraud Detection using Data Engineering Techniques:

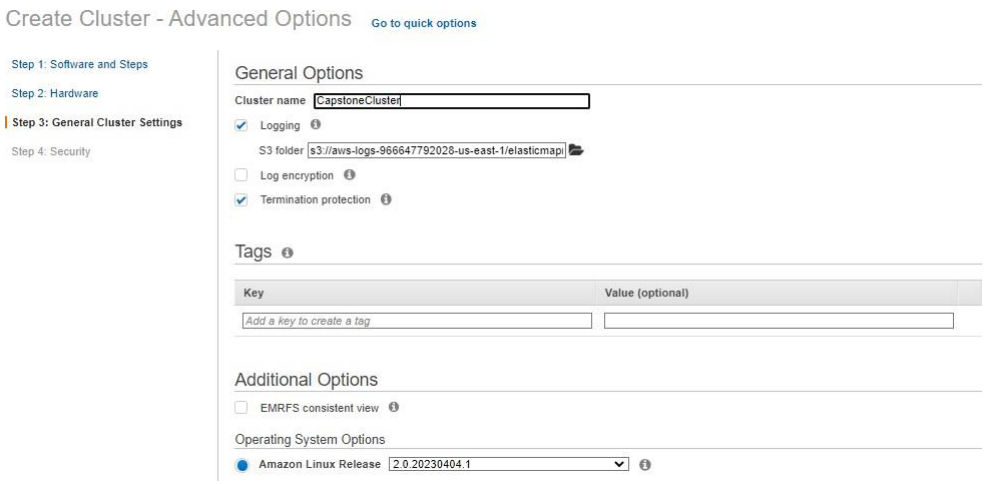
Following are the Steps, Commands and Screenshots of the entire execution:

1. The necessary files ‘card_member.csv’, ‘member_score.csv’, ‘uszipsv.csv’ and ‘card_transactions.csv’ are uploaded into the AWS S3 Bucket as shown below:

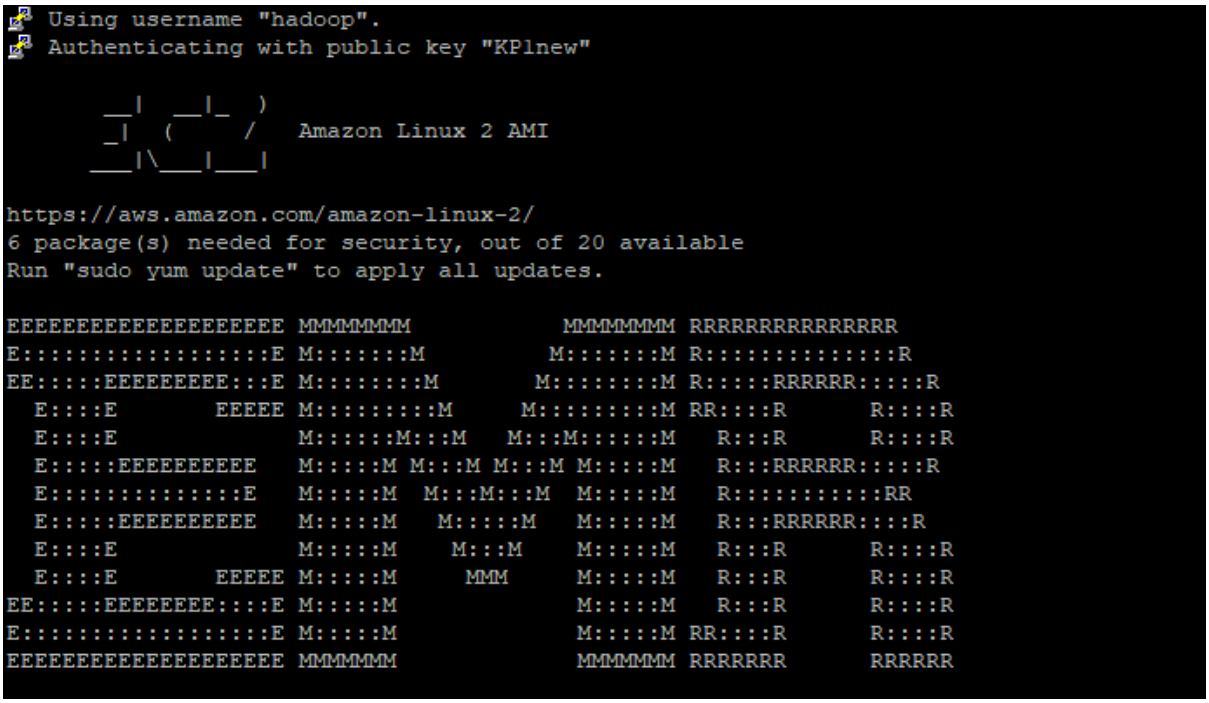


2. Now, start the EMR Cluster in PuTTy with the selections shown in the below screenshots (Same steps to start the EMR have been covered in LoadNoSQL file) :





3. Once configured, the terminal opens as shown below:



4. Now, execute the following commands one by one in Hadoop to copy the files from S3 bucket:

   Commands:
   aws s3 cp s3://capstoneprojectakash/card_member.csv .

   aws s3 cp s3://capstoneprojectakash/member_score.csv .

   aws s3 cp s3://capstoneprojectakash/card_transactions.csv .

   Output Screenshot:

```
[hadoop@ip-172-31-93-134 ~]$ aws s3 cp s3://capstoneprojectakash/card_member.csv .
download: s3://capstoneprojectakash/card_member.csv to ./card_member.csv
[hadoop@ip-172-31-93-134 ~]$ aws s3 cp s3://capstoneprojectakash/member_score.csv .
download: s3://capstoneprojectakash/member_score.csv to ./member_score.csv
[hadoop@ip-172-31-93-134 ~]$ aws s3 cp s3://capstoneprojectakash/card_transactions.csv .
download: s3://capstoneprojectakash/card_transactions.csv to ./card_transactions.csv
[hadoop@ip-172-31-93-134 ~]$
```

5. Start hive and create a database and use it by following the below commands:

   hive
   create database ccfd_capstone;
   use ccfd_capstone;

   OUTPUT:

```
[hadoop@ip-172-31-93-134 ~]$ hive
Hive Session ID = 9134bb40-5a37-4681-b042-109964ecdb58

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: false
hive> create database ccfd_capstone;
OK
Time taken: 2.152 seconds
hive> use ccfd_capstone;
OK
Time taken: 0.094 seconds
```

6. Use the following command to create a table named card_member:

   create table if not exists card_member(card_id bigint,member_id bigint,member_joining_dt string,card_purchase_dt string,country string,city string) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile tblproperties("skip.header.line.count"="1");

   OUTPUT:

```
hive> create table if not exists card_member(card_id bigint,member_id bigint,member_joining_dt string,card_purchase_dt string,country string,city string) row format del
imited fields terminated by ',' lines terminated by '\n' stored as textfile tblproperties("skip.header.line.count"="1");
OK
Time taken: 0.697 seconds
```

7. Then, use the below command to copy the data from card_member.csv file to the card_member table:
   load data local inpath '/home/hadoop/card_member.csv' into table card_member;

   OUTPUT:

```
hive> load data local inpath '/home/hadoop/card_member.csv' into table card_member;
Loading data to table ccfd_capstone.card_member
OK
Time taken: 1.094 seconds
```

8. Similarly use the following commands one by one to create a table for member_score.csv file and load the data from the file into the table:

   create table if not exists member_score(member_id bigint,score int) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile tblproperties("skip.header.line.count"="1");

load data local inpath '/home/hadoop/member_score.csv' into table member_score;

OUTPUT:

```
hive> create table if not exists member_score(member_id bigint,score int) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile tblp
roperties("skip.header.line.count"="1");
OK
Time taken: 0.129 seconds
hive>
    > load data local inpath '/home/hadoop/member_score.csv' into table member_score;
Loading data to table ccfd_capstone.member_score
OK
Time taken: 0.855 seconds
```

9. Now, we can test if the data in the files are loaded properly into the tables. Both files has 999 rows each excluding the header. The below commands can be run one by one to find if all the rows have been loaded into the tables:

   select count(*) from card_member;

   select count(*) from member_score;

   OUTPUT:

```
hive> select count(*) from card_member;
Query ID = hadoop_20230530053710_53cfd2f0-5ad2-492d-a17d-9bfb21210f2b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1685424166206_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 7.58 s
--------------------------------------------------------------------------------
OK
999
Time taken: 102.769 seconds, Fetched: 1 row(s)
hive> select count(*) from member_score;
Query ID = hadoop_20230530053853_1d2620c9-e764-41a7-a81b-00d8448138d7
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1685424166206_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 0.44 s
--------------------------------------------------------------------------------
OK
999
Time taken: 1.291 seconds, Fetched: 1 row(s)
```

10. In the Hadoop EMR instance, type the below command to create a repository to download the MongoDB files:

    #To come out of Hive and enter Hadoop
    cd

#To mention the URL for mongodb installation

sudo vi /etc/yum.repos.d/mongodb-org-6.0.repo

Then copy and paste the below code:

```
[mongodb-org-6.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2/mongodb-org/6.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-6.0.asc
```

And then type the below to save it
:w!

And type the below code to quit it

:q

Now, type the below command to install the latest stable version of MongoDB:

sudo yum install -y mongodb-org

11. Now, you can use the below code to point out to the file that contains the MongoDB Repository and add the below code:

#To go to the respective directory of mongodb
cd /etc/yum.repos.d

#To ensure the official version of Mongodb, open the file using the below command
vi mongodb-org-6.0.repo

The file contains the below details. This is to ensure that mongodb has been officially downloaded:
[mongodb-org-6.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2/mongodb-org/6.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-6.0.asc

hadoop@ip-172-31-93-134:/etc/yum.repos.d

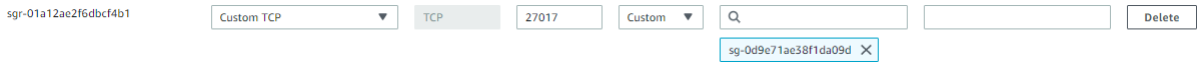
```
[mongodb-org-6.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2/mongodb-org/6.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-6.0.asc

~
~
~
~
~
~
~
```

12. Install mongodb-org using the below code in the above path:
    With the below code, change to root user:
    sudo su
    Now, open /etc/mongod.conf using the below codes:
    cd /etc
    vi mongod.conf

13. Now, set the bindIP to 0.0.0.0

```
root@ip-172-31-93-134:/etc
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true
#  engine:
#  wiredTiger:

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0  # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6 addresses or, alternatively, use the net.bindIpAll setting.


#security:

#operationProfiling:

#replication:

#sharding:

## Enterprise-Only Options

#auditLog:

#snmp:
~
"mongod.conf" 42L, 719B
```

14. Now, go to Master Security group of EMR cluster and add an inbound rule to allow traffic on port 27017 with source as the EMR slave node security group as shown below:



15. Now, go back to hadoop using the below code to move to hadoop user:

    sudo su hadoop

    #To come out of root folder

    cd

16. Now, use the below code to start the mongod service

    sudo service mongod start

```
[hadoop@ip-172-31-93-134 ~]$ sudo service mongod start
Redirecting to /bin/systemctl start mongod.service
```

17. And use the below code to check the mongod status

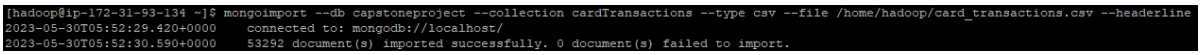
    sudo service mongod status

```
[hadoop@ip-172-31-93-134 ~]$ sudo service mongod status
Redirecting to /bin/systemctl status mongod.service
● mongod.service - MongoDB Database Server
   Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2023-05-30 05:51:18 UTC; 13s ago
     Docs: https://docs.mongodb.org/manual
 Main PID: 15178 (mongod)
    Tasks: 34
   Memory: 65.4M
   CGroup: /system.slice/mongod.service
           └─15178 /usr/bin/mongod -f /etc/mongod.conf

May 30 05:51:18 ip-172-31-93-134 systemd[1]: Started MongoDB Database Server.
May 30 05:51:18 ip-172-31-93-134 mongod[15178]: {"t":{"$date":"2023-05-30T05:51:18.037Z"},"s":"I",  "c":"CONTROL",  "id":7484500, "ctx":"-","msg":"Environme...o false"}
Hint: Some lines were ellipsized, use -l to show in full.
```

18. Now, use the below code to import the file into MongoDB. You can notice that all 53292 rows in the file are copied as documents in the MongoDB

    mongoimport --db capstoneproject --collection cardTransactions --type csv --file /home/hadoop/card_transactions.csv --headerline

    OUTPUT:



19. Use the below code to start pyspark to define the tables to be read and to be written back

```
pyspark --conf
"spark.mongodb.read.connection.uri=mongodb://172.31.93.134:27017/capstoneproject.cardTransac
tions?readPreference=primaryPreferred"          --conf
"spark.mongodb.write.connection.uri=mongodb://172.31.93.134:27017/capstoneproject.lookupTran
s"          --packages org.mongodb.spark:mongo-spark-connector_2.12:10.1.1
```

20. Use the below commands in order to create the lookup table and write the data into lookupTrans table:

```
#Importing the necessary pyspark packages as mentioned below to create the lookup table

from pyspark.sql import SparkSession

from pyspark.sql.functions import *

from pyspark.sql.types import *

from pyspark.sql.window import Window


#To define the file system path

from os.path import abspath

warehouse_location = abspath('spark-warehouse')

#To define the spark app

spark =(SparkSession.builder.appName("Integration_Mongo_Hive") \

    .master("local[*]") \

    .config("spark.sql.warehouse.dir", warehouse_location) \

    .enableHiveSupport() \

    .getOrCreate())

#To read the mongodb format file from the spark warehouse

ct_df=spark.read.format("mongodb").load()
```

#To drop “_id” column and change the data type of the “transaction_dt” column

```
ct_df2=ct_df.drop("_id").withColumn("transaction_dt",to_timestamp('transaction_dt','dd-MM-yyyy HH:mm:ss'))
```

#To arrange the card_id in descending order by defining the Window Function

```
windowSpec = Window\
        .partitionBy("card_id")\
        .orderBy(desc("transaction_dt"))
```

#To filter only GENUINE transactions and apply the Window function

```
ct_df3=ct_df2.filter(col('status')=='GENUINE').withColumn("row_number",row_number().over(windowSpec))
```

#To filter and consider only upto 10 rows

```
ct_df4 = ct_df3.filter(col('row_number')<=10)
```

#To create the temporary view ‘v_card_transactions’

```
ct_df4.createOrReplaceTempView('v_card_transactions')
```

#To write the spark sql query to find UCL value, take the last postcode and the last transaction date

```
ct_df5=spark.sql('select card_id,member_id,max(postcode) as postcode,max(transaction_dt) as transaction_dt,avg(amount) as avg_amount,stddev(amount) as std_dev_amount,(avg(amount)+3*stddev(amount)) as UCL from v_card_transactions group by card_id,member_id')
```

#To read the data from the card_member and member_score tables from Hive and filter out the non-null values

```
card_member_df=spark.read.table("ccfd_capstone.card_member")

member_score_df=spark.read.table("ccfd_capstone.member_score")

member_score_df=member_score_df.filter(member_score_df.member_id.isNotNull() & member_score_df.score.isNotNull())

card_member_df=card_member_df.filter( card_member_df.card_id.isNotNull() & card_member_df.member_id.isNotNull())
```

#To make an inner join between the card transactions and member score table

```
ct_df6=ct_df5.join(member_score_df,"member_id","inner")

ct_df6.write.format("mongodb").mode("append").save()
```

Output:

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.1-amzn-0
      /_/

Using Python version 3.7.16 (default, Mar 10 2023 03:25:26)
Spark context Web UI available at http://ip-172-31-93-134.ec2.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1685424166206_0005).
SparkSession available as 'spark'.
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import *
>>> from pyspark.sql.types import *
>>> from pyspark.sql.window import Window
>>> from os.path import abspath
>>> warehouse_location = abspath('spark-warehouse')
>>> spark =(SparkSession.builder.appName("Integration_Mongo_Hive") \
...         .master("local[*]") \
...         .config("spark.sql.warehouse.dir", warehouse_location) \
...         .enableHiveSupport() \
...         .getOrCreate())
23/05/30 05:54:48 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
>>> ct_df=spark.read.format("mongodb").load()
>>> ct_df2=ct_df.drop("_id").withColumn("transaction_dt",to_timestamp('transaction_dt','dd-MM-yyyy HH:mm:ss'))
>>> windowSpec = Window\
...             .partitionBy("card_id")\
...             .orderBy(desc("transaction_dt"))
>>> ct_df3=ct_df2.filter(col('status')=='GENUINE').withColumn("row_number",row_number().over(windowSpec))
>>> ct_df4 = ct_df3.filter(col('row_number')<=10)
```

```
>>> ct_df4.createOrReplaceTempView('v_card_transactions')
>>> ct_df5=spark.sql('select card_id,member_id,max(postcode) as postcode,max(transaction_dt) as transaction_dt,avg(amount) as avg_amount,stddev(amount) as std_dev_amoun
t,(avg(amount)+3*stddev(amount)) as UCL from v_card_transactions group by card_id,member_id')
>>> card_member_df=spark.read.table("ccfd_capstone.card_member")
member_score_df=spark.read.table("ccfd_capstone.member_score")
member_score_df=member_score_df.filter(member_score_df.member_id.isNotNull() & member_score_df.score.isNotNull())
card_member_df=card_member_df.filter( card_member_df.card_id.isNotNull() &  card_member_df.member_id.isNotNull())
23/05/30 05:57:21 INFO HiveConf: Found configuration file file:/etc/spark/conf.dist/hive-site.xml
23/05/30 05:57:21 WARN HiveConf: HiveConf of name hive.server2.thrift.url does not exist
23/05/30 05:57:21 INFO metastore: Trying to connect to metastore with URI thrift://ip-172-31-93-134.ec2.internal:9083
23/05/30 05:57:21 INFO metastore: Opened a connection to metastore, current connections: 1
23/05/30 05:57:21 INFO metastore: Connected to metastore.
>>> member_score_df=spark.read.table("ccfd_capstone.member_score")
>>> member_score_df=member_score_df.filter(member_score_df.member_id.isNotNull() & member_score_df.score.isNotNull())
>>> card_member_df=card_member_df.filter( card_member_df.card_id.isNotNull() &  card_member_df.member_id.isNotNull())
>>> ct_df6=ct_df5.join(member_score_df,"member_id","inner")
>>> ct_df6.write.format("mongodb").mode("append").save()
>>> exit()
```

21. In order to check if the collections are created in MongoDB, use the below code:

#To start mongosh

mongosh

#To choose the database to work on

use capstoneproject

show collections

Output:

```
[hadoop@ip-172-31-93-134 ~]$ mongosh
Current Mongosh Log ID: 647590d023ba45ea1e0d1a53
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.9.1
Using MongoDB:          6.0.6
Using Mongosh:          1.9.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2023-05-30T05:51:19.001+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2023-05-30T05:51:19.001+00:00: vm.max_map_count is too low
------

test> use capstoneproject
switched to db capstoneproject
capstoneproject> show collections
cardTransactions
lookupTrans
```

22. Now, count the number of records in each table

db.cardTransactions.count()

db.lookupTrans.count()

OUTPUT:

```
capstoneproject> db.cardTransactions.count()
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
53292
capstoneproject> db.lookupTrans.count()
999
```

23. To ensure that the records are loaded, use the below command:

db.lookupTrans.find()

OUTPUT:

```
capstoneproject> db.lookupTrans.find()
[
  {
    _id: ObjectId("64759065f34fb77e3102d6f8"),
    member_id: Long("9250698176266"),
    card_id: Long("340028465709212"),
    postcode: 95636,
    transaction_dt: ISODate("2018-01-02T03:25:35.000Z"),
    avg_amount: 6863758.9,
    std_dev_amount: 3326644.6481975215,
    UCL: 16843692.844592564,
    score: 233
  },
  {
    _id: ObjectId("64759065f34fb77e3102d6f9"),
    member_id: Long("835873341185231"),
    card_id: Long("340054675199675"),
    postcode: 98395,
    transaction_dt: ISODate("2018-01-15T19:43:23.000Z"),
    avg_amount: 4976333.7,
    std_dev_amount: 3225433.9971526675,
    UCL: 14652635.691458002,
    score: 631
  },
  {
    _id: ObjectId("64759065f34fb77e3102d6fa"),
```

Now, you can go back to Hadoop using the exit() command.

So, by following the above steps the Lookup table has been created.

LogicFinal

1. Install the packages using the codes mentioned below:

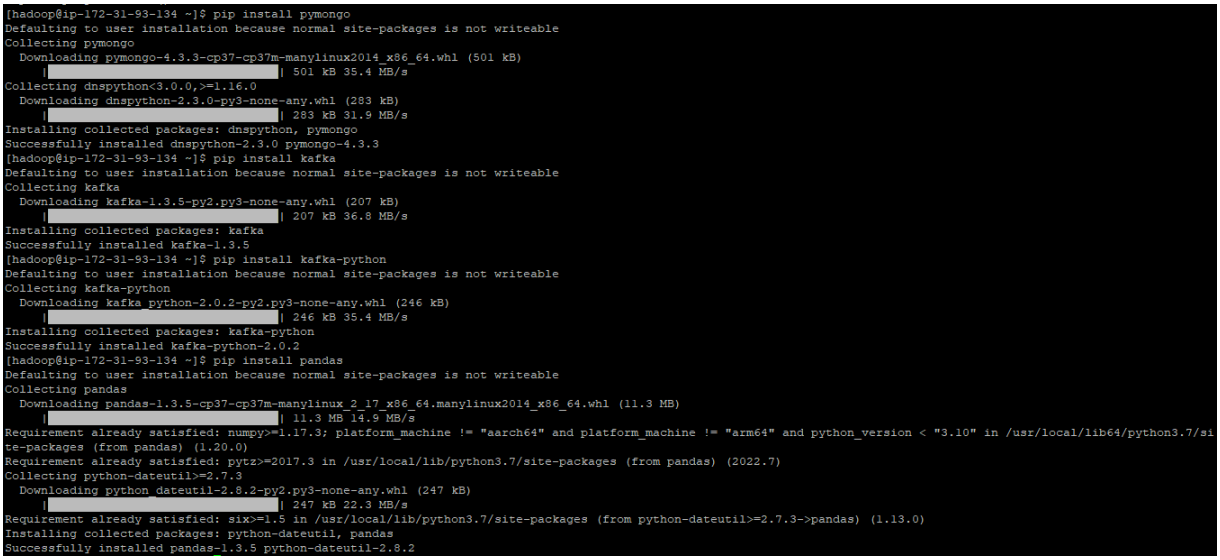
pip install pymongo

pip install kafka

pip install kafka-python

pip install pandas

OUTPUT:



Now, as per the below guidelines, the directories are created

- A directory named "**python**" should be created.
- It should contain a directory named "**src**".
- The "**src**" directory should have two directories named "**db**" and "**rules**".
- The "**src**" directory should have a python file named "**driver.py**" which should be the calling the other files and should be the entry point of your code.
- The "**rules**" directory should contain a "**rules.py**" file where you write the functions to check for the three rules mention earlier. The "**db**" directory should have the "**geo_map.py**" and "**dao.py**" along with the **uszipsv.csv.**
- The **driver.py** file should contain the code to read the messages from Kafka and call necessary functions from the python files present in the "**rules**" and "**db**" directory to classify the incoming transaction as fraud or genuine.

Code:

#Creation of the python directory

mkdir python

#Navigate to the python directory

cd python

#Creation of src directory

mkdir src

#Navigate to src directory

cd src

#Creation of db and rules directories

mkdir db

mkdir rules

#Navigation to rules directory

cd rules

vi rules.py

Type the below code in rules.py

```
import sys

sys.path.append("..")

from db.geo_map import GEO_Map

from db.dao import *

from datetime import datetime

from datetime import timedelta


#To read the transaction data from the lookup table in mongodb, which has been populated already by the batch load

collection_lookup=readMongoLookup()


#In order to validate if the transaction is fraud or genuine, driver.py file would iterate through the processMessage function as shown below

def processMessage(msg):


    #To receive the card ID of the current transaction

    card_id=msg['card_id']


    #To create an empty dictionary to populate the data

    document={}


    #To get the card id and it's respective parameters from historical data

    cursor = collection_lookup.find({"card_id":card_id})
```

```
    #To get the historical transaction of the card id
    for document in cursor:
        document=document

    #To pass the data of the Card to validate the transaction
    status = authoriseTransaction(msg, document)
    return status



#To validate the transaction
def authoriseTransaction(transaction_data, lookup_data):
    auth=""

    #To check if current transaction amount is less than  the Upper Credit Limit (UCL) and confirm if the transaction is GENUINE or FRAUD
    if transaction_data['amount'] <= lookup_data['UCL']:
        auth="GENUINE"
    else:
        auth="FRAUD"
    #To display the message as FRAUD for UCL less than 200
    if lookup_data["score"]<200:
        auth="FRAUD"

    #To determine the time gap between two transactions using the formula of speed
    dist=calcDist(transaction_data, lookup_data)
    time=calcTime(transaction_data, lookup_data)
    speed= dist/time

    #To declare that the transaction if fraudulent if the speed is greater than 900 KM/Hr
```

```
    if speed>900:
        auth ="FRAUD"
    return auth


#To calculate the distance between the locations of last transaction and the current transaction, the below function is defined (For Speed Formula)
def calcDist(transaction_data, lookup_data):
    geo_map = GEO_Map.get_instance()
    DATE_FORMAT = '%d-%m-%Y %H:%M:%S'

    #To get the latitude and longitude of current transaction
    current_lat, current_long = geo_map.get_lat(str(transaction_data['postcode'])).iloc[0], geo_map.get_long(str(transaction_data['postcode'])).iloc[0]

    #To get the latitude and longitude of the last transaction
    last_lat, last_long = geo_map.get_lat(str(lookup_data['postcode'])).iloc[0], geo_map.get_long(str(lookup_data['postcode'])).iloc[0]

    #To calculate the distance between the locations of last transaction and the current transaction
    dist = geo_map.distance(current_lat, current_long, last_lat, last_long)
    return dist

#To calculate the duration between the locations of the last transaction and the current transaction (For Speed Formula)
def calcTime(transaction_data, lookup_data):

    #To get the transaction date and time of the current transaction
    transaction_dt_str = transaction_data['transaction_dt']

    #To convert to date format from string
    transaction_dt = datetime.strptime(transaction_dt_str, '%d-%m-%Y %H:%M:%S')
```

```
    #To get the last transaction's transaction date and subtract it from the current transaction date
    time = transaction_dt - lookup_data['transaction_dt']

    #To convert the time difference into hours
    hours = time.total_seconds() / 3600
    return hours
```

Now, use **:wq!** to save the file and quit the editor mode and then, you can type the below codes:

#To go to Hadoop folder and then go to the db directory

cd

cd python

cd src

cd db

OUTPUT:

```
[hadoop@ip-172-31-93-134 ~]$ mkdir python
[hadoop@ip-172-31-93-134 ~]$ cd python
[hadoop@ip-172-31-93-134 python]$ mkdir src
[hadoop@ip-172-31-93-134 python]$ cd src
[hadoop@ip-172-31-93-134 src]$ mkdir db
[hadoop@ip-172-31-93-134 src]$ mkdir rules
[hadoop@ip-172-31-93-134 src]$ cd rules
[hadoop@ip-172-31-93-134 rules]$ vi rules.py
[hadoop@ip-172-31-93-134 rules]$ cd
[hadoop@ip-172-31-93-134 ~]$ cd python
[hadoop@ip-172-31-93-134 python]$ cd src
[hadoop@ip-172-31-93-134 src]$ cd db
```

#To copy the uszipsv.csv file from S3

aws s3 cp s3://capstoneprojectakash/uszipsv.csv .

OUTPUT:

```
[hadoop@ip-172-31-93-134 db]$ aws s3 cp s3://capstoneprojectakash/uszipsv.csv .
download: s3://capstoneprojectakash/uszipsv.csv to ./uszipsv.csv
```

#To create the geo_map file

vi geo_map.py

Type the below code inside the geo_map file

```
import math
import pandas as pd


class GEO_Map():
        """
        It hold the  map for zip code and its latitute and longitute
        """
        __instance = None

        @staticmethod
        def get_instance():
                """ Static access method. """
                if GEO_Map.__instance == None:
                        GEO_Map()
                return GEO_Map.__instance

        def __init__(self):
                """ Virtually private constructor. """
                if GEO_Map.__instance != None:
                        raise Exception("This class is a singleton!")
                else:
                        GEO_Map.__instance = self
                        self.map = pd.read_csv("/home/hadoop/python/src/db/uszipsv.csv",
header=None, names=['A',"B",'C','D','E'])
                        self.map['A'] =  self.map['A'].astype(str)

        def get_lat(self, pos_id):
                return self.map[self.map.A == pos_id ].B
```

```
        def get_long(self, pos_id):
                return self.map[self.map.A == pos_id ].C

        def distance(self, lat1, long1, lat2, long2):
                theta = long1 - long2
                dist = math.sin(self.deg2rad(lat1)) * math.sin(self.deg2rad(lat2)) +
math.cos(self.deg2rad(lat1)) * math.cos(self.deg2rad(lat2)) * math.cos(self.deg2rad(theta))
                dist = math.acos(dist)
                dist = self.rad2deg(dist)
                dist = dist * 60 * 1.1515 * 1.609344
                return dist

        def rad2deg(self, rad):
                return rad * 180.0 / math.pi

        def deg2rad(self, deg):
                return deg * math.pi / 180.0
```

Also create the below file named dao.py:

vi dao.py

Type the below code inside the file:

```
from pymongo import MongoClient
from kafka import KafkaConsumer
from json import loads, dumps


#To define the Mongo Client
client = MongoClient('localhost', 27017)

#To mention the Database name inside the Mongo Client
db = client['capstoneproject']
```

```
#Function definition to read from Kafka
def readKafka():
    consumer = KafkaConsumer(
        'transactions-topic-verified',
        bootstrap_servers=['18.211.252.152:9092'],
        auto_offset_reset='earliest',
        enable_auto_commit=True,
        value_deserializer=lambda x: loads(x.decode('utf-8')))

    return consumer



#To read the lookup_transaction data from MongoDB
def readMongoLookup():
    try:
        lookup_transaction=db.lookupTrans
        print("Connection Successful")
    except:
        print("Connection to MongoDB is unsuccessful")

    return lookup_transaction



#To write back the FRAUDULENT and GENUINE Transactions into MongoDB
def writeToMongo(transaction):
    collection_name = db.cardTransactions
    rec_id = collection_name.insert_one(transaction)
    print("The data is inserted with Record ID",rec_id)
```

OUTPUT:

After creating the directories and files, type the below commands to go the src directory and to create the driver.py file. It should be the entry point of our code

#To navigate to the src directory

cd python/src/

#To create the driver.py file

vi driver.py

Now, type the following inside driver.py file:

#To import the rules.py file from the rules folder

from rules.rules import processMessage

#To import the dao.py file from the db folder

from db.dao import *

if __name___== "__main__":

#To read the data from Kafka Topic

consumer = readKafka()

#To loop through the Kafka message

for message in consumer:

#To get actual message

currentMessage = message.value

#To determine if the message that has been received (transaction) is fraudulent or not

status = processMessage(currentMessage)

#To push the message status into the dictionary

currentMessage["status"]=status

#To print the current message status

print(currentMessage)

#To write the message in MongoDB

writeToMongo(currentMessage)

Screenshots:

```
[hadoop@ip-172-31-93-134 ~]$ cd python/src/
[hadoop@ip-172-31-93-134 src]$ vi driver.py
```

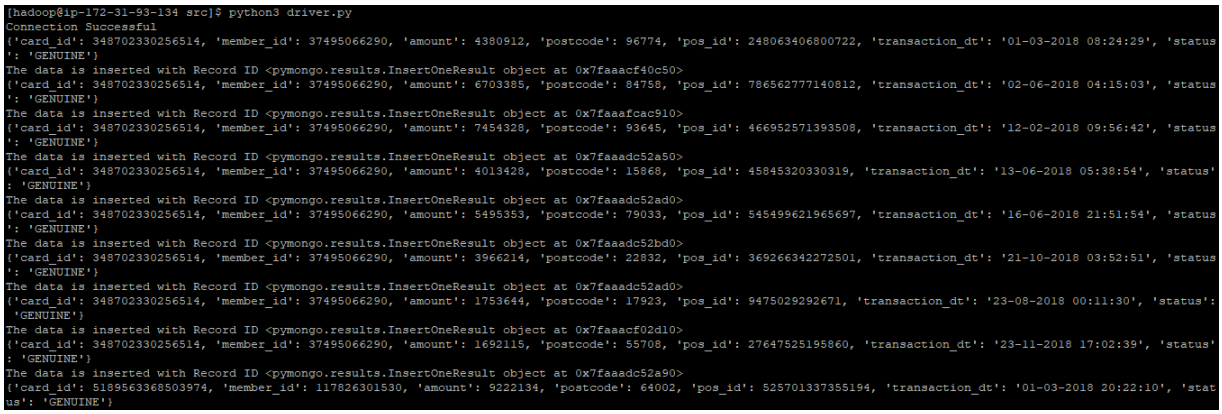
hadoop@ip-172-31-93-134:~/python/src

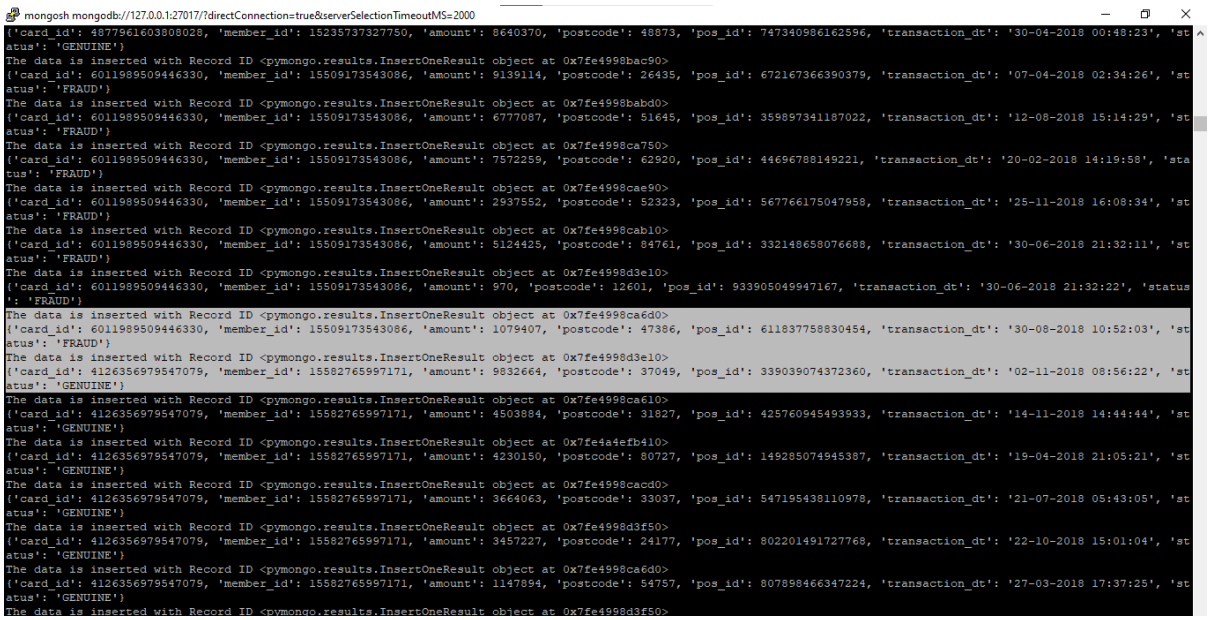
```
#To import the rules.py file from the rules folder
from rules.rules import processMessage

#To import the dao.py file from the db folder
from db.dao import *


if __name__ == "__main__":
    #To read the data from Kafka Topic
    consumer = readKafka()

    #To loop through the Kafka message
    for message in consumer:

        #To get actual message
        currentMessage = message.value

        #To determine if the message that has been received (transaction) is fraudulent or not
        status = processMessage(currentMessage)

        #To push the message status into the dictionary
        currentMessage["status"]=status

    #To print the current message status
        print(currentMessage)

        #To write the message in MongoDB
        writeToMongo(currentMessage)
~
```

After creating the directories, type the below command to run the program

python3 driver.py

OUTPUT:



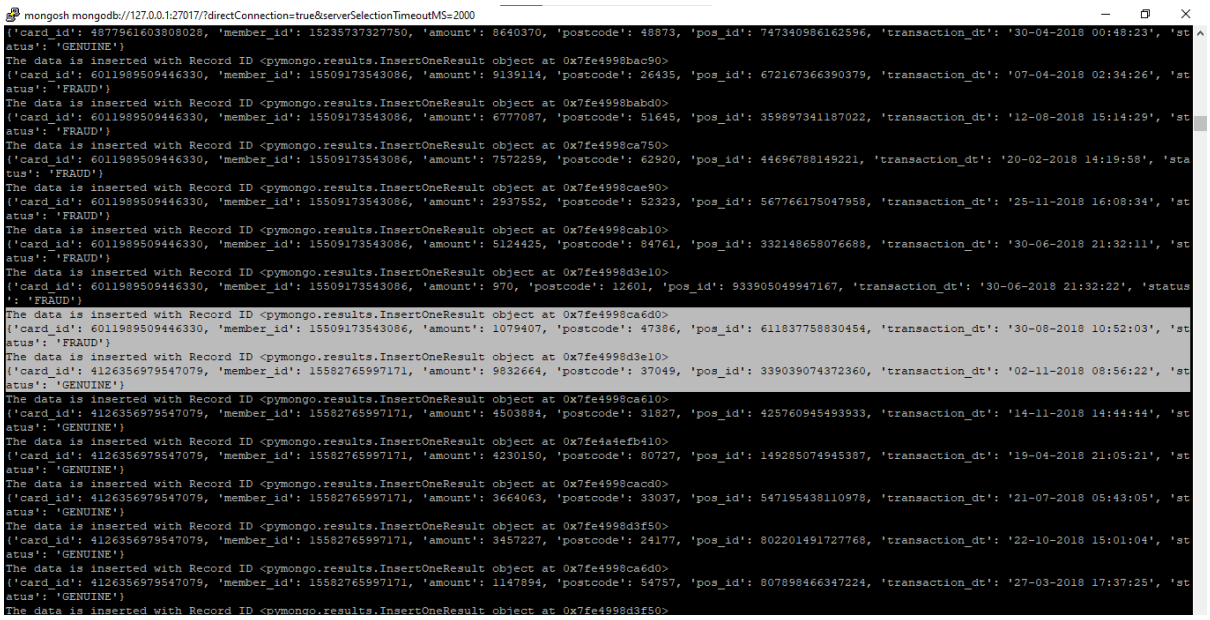


From the above screenshots, you can notice that the transactions have been classified as GENUINE or FRAUD and they get populated in the lookupTrans collection in MongoDB

We can use Ctrl+C to stop the program

We can validate the same as shown below:

Take two records, one for GENUINE and one for FRAUDULENT as shown in the below screenshot:



CASE 1

Output:

The data is inserted with Record ID <pymongo.results.InsertOneResult object at 0x7fe4998ca6d0>

{'card_id': 6011989509446330, 'member_id': 15509173543086, 'amount': 1079407, 'postcode': 47386, 'pos_id': 611837758830454, 'transaction_dt': '30-08-2018 10:52:03', 'status': 'FRAUD'}

Solution:

From the member_score.csv file, we can notice that the score of the member is 176, which is less than 200. So, it is confirmed that the Credit score of the member is less than the threshold value and so, the output has appeared as ‘FRAUD’ for this transaction.

CASE 2

Output:

The data is inserted with Record ID <pymongo.results.InsertOneResult object at 0x7fe4998d3e10>

{'card_id': 4126356979547079, 'member_id': 15582765997171, 'amount': 9832664, 'postcode': 37049, 'pos_id': 339039074372360, 'transaction_dt': '02-11-2018 08:56:22', 'status': 'GENUINE'}

Solution:

Since this has passed the UCL, Credit Score and ZIP code Analysis checks, this transaction has been termed as ‘GENUINE’