Apache Kafka – Spark Integration Using PySpark

Table of Contents

ASSIGNMENT II APACHE KAFKA – SPARK INTEGRATION 1

GENERAL ASSUMPTIONS: 3
ARCHITECTURE 3
EXERCISE 1 3
Steps and Results for Exercise 1(a): Installing Kafka 3
Steps for Exercise 1(b): Send data from Excel to Kafka 5
For Exercise 1(c): For reading of messages from Kafka Topics and Processing in Spark 6
Steps and Results for Exercise 1(d): Create a Kafka topic and sending all the data into a topic 7
EXERCISE 2: READ DATASET AND DEVELOP A MACHINE LEARNING MODEL TO DETECT THE CUSTOMER IS GOOD OR BAD 10
1. Random forest classifier: 11
2. Extreme gradient boosted (XGBoost) classifier: 12
EXERCISE 3: STREAM DATA PROCESSING FROM KAFKA AND SENDING ALERTS TO ANOTHER KAFKA 13
Reading messages in Kafka 13
Pre-processing data after reading from Kafka 13
Processing Kafka data in Python with Machine Learning Model 15
Sending Classified data for Kafka Alert Topic 17

General Assumptions:

- Customer transaction format assumed was same as the Kaggle dataset specified.
- There is no need for separate join with Master data since Customer master data was not available for the given dataset.
- There is no need to do SMS integration since it is out of scope for this course.
- Final fraud detection data and non-fraud data will be segregated and sent to different Kafka topic for consumption of application to alert.

Architecture

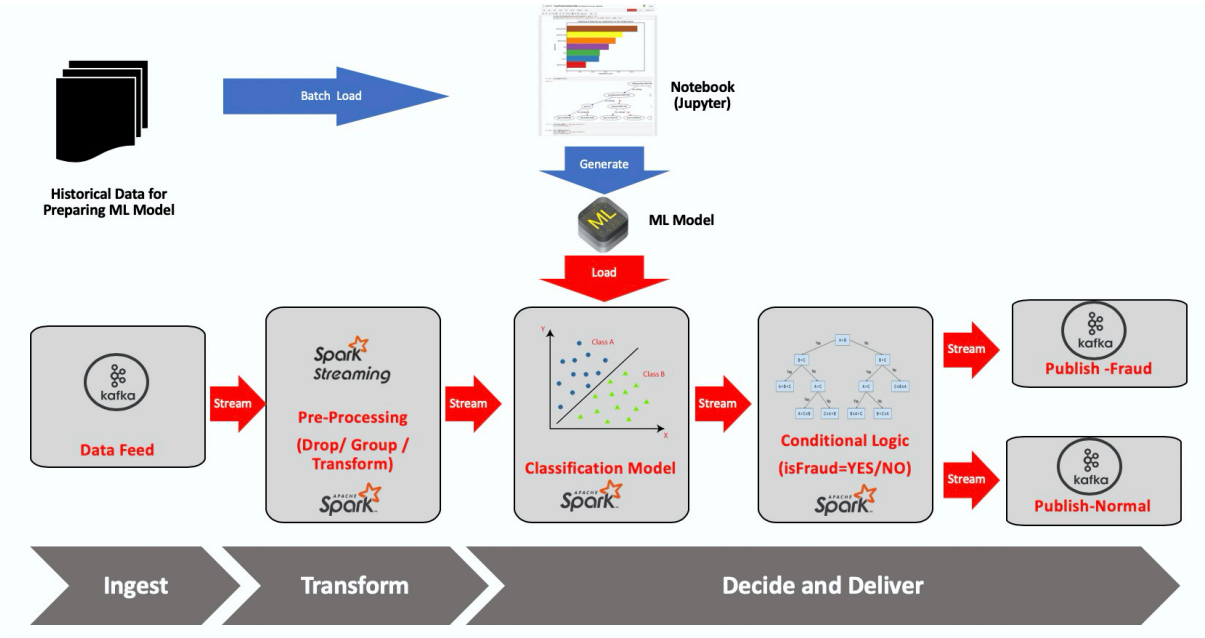


Fig 1: Architecture Diagram

Exercise 1

Scope:

- Install & Setup Kafka,
- Send and Store all the transaction data in Kafka from an Excel file using Python
- Read data from a Kafka topic using Python
- Install, Setup PySpark and test its functioniong

Steps and Results for Exercise 1(a): Installing Kafka

1. Download and Install Kafka as mentioned here https://kafka.apache.org/quickstart
2. cd {$KAFKA_INSTALLDIR}
3. Start Zookeeper (in a separate console)

```
> bin/zookeeper-server-start.sh config/zookeeper.properties
```

Result:

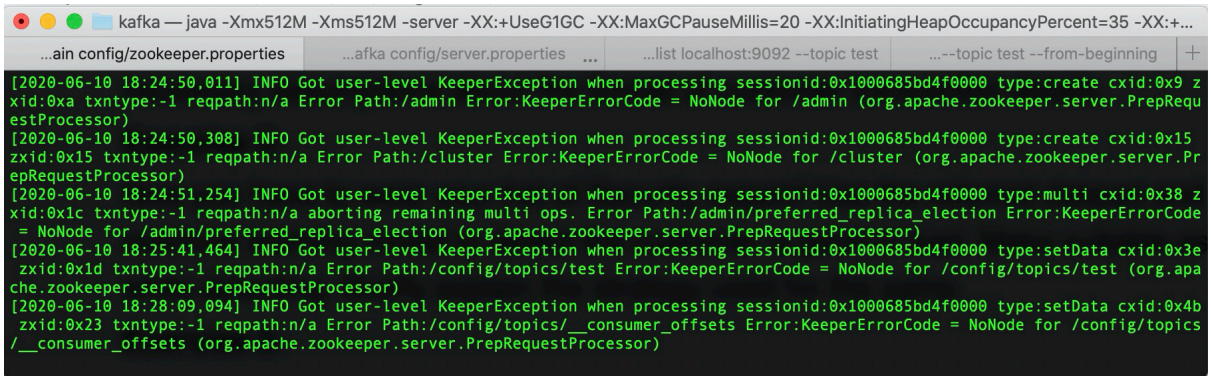


Fig 2: Starting Zookeeper

4. Start Kafka Server (in a separate console)

```
>bin/kafka-server-start.sh config/server.properties
```

Result:

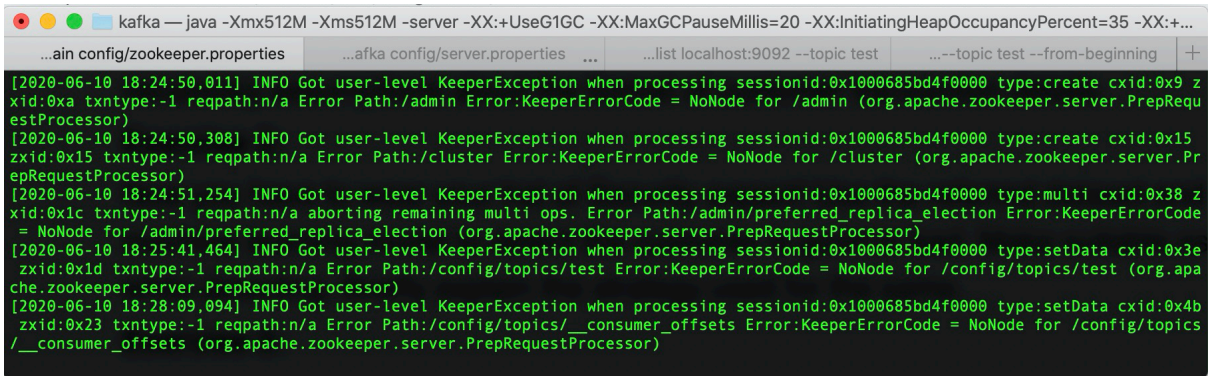


Fig 3: Starting Kafka Server

5. Create a Topic (in a separate console)

```
>bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --
replication-factor 1 --partitions 1 --topic test
```

6. Check the Topic

```
>bin/kafka-topics.sh --list --broker-list localhost:9092
```

Note: --bootstrap-server did not work on earlier versions, so used -brokerlist

7. Send messages to the created topic

```
>bin/kafka-console-producer.sh --bootstrap-server localhost:9092
--topic test
```

Result for steps 5,6,&7:

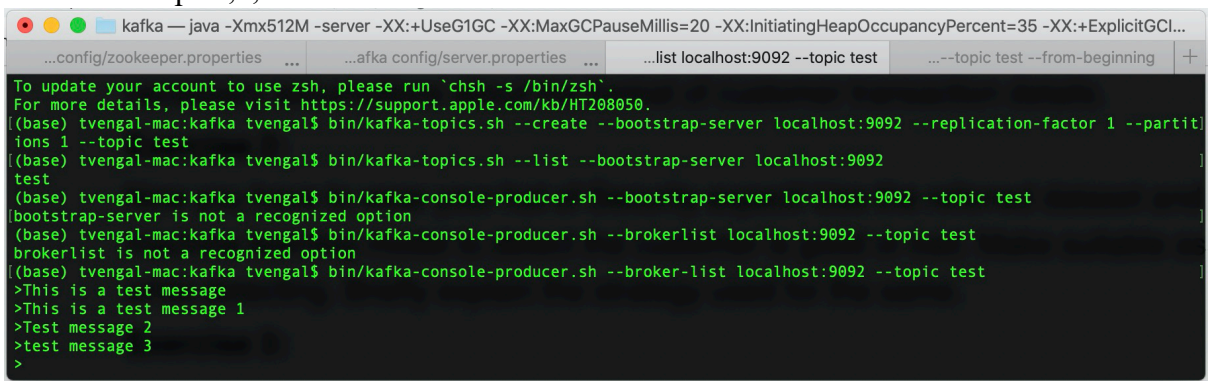


Fig 4: Sending a test message in Kafka Producer

8. Start a Consumer and validate the topic

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
--topic test --from-beginning
```

Result:

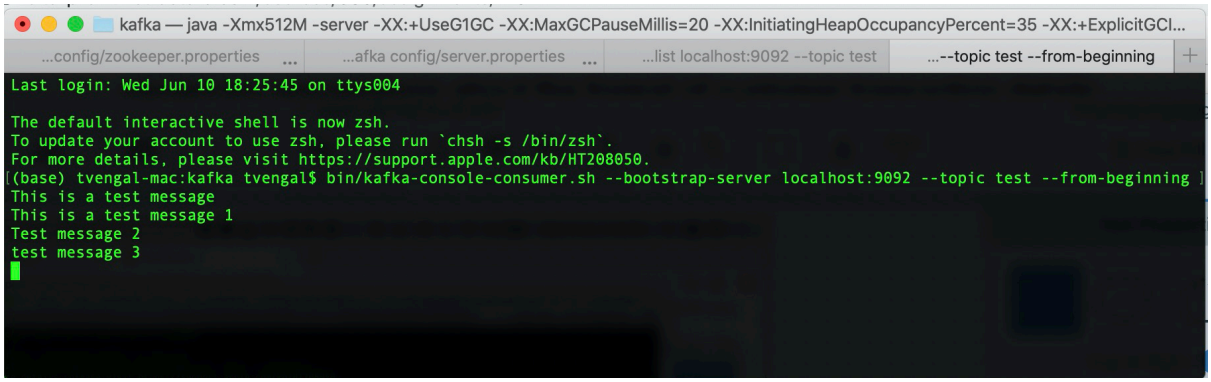


Fig 4: Receiving messages in Kafka Consumer client

Steps for Exercise 1(b): Send data from Excel to Kafka

- Import Python-Kafka library using the following command

  ```
  >pip install kafka-python
  ```

- Run the program as given in the <fill>.ipynb
- This has 3 ways of sending
  1. Generate numeric messages and send in a loop
  2. Sending in Json format without header
  3. Sending in Json format with headers
- We chose to use the option (3) sending in Json format with headers

For Exercise 1(c): For reading of messages from Kafka Topics and Processing in Spark

1. Downloaded data set from [https://www.kaggle.com/ntnu-testimon/paysim1 (Links to an external site.)](https://www.kaggle.com/ntnu-testimon/paysim1)
2. Tried the following options
    a. Downloaded Hortonworks VM from Cloudera website, which was over 20GB. But could not install since it required more than 100GB of free space on laptop.
    b. Looked for an alternate method to install manually Hadoop and Spark, but was had issues on Macbook.
    c. Installed pyspark using

    ```
    >pip install pyspark
    ```

    Pyspark installation worked and this method was chosen to proceed.
3. Set the following environment variables

    (Code: run bash_profile.txt)

    a. PATH
    b. CLASSPATH
    c. SPARK_HOME
    d. PYSPARK_DRIVER_PYTHON=jupyter
    e. PYSPARK_DRIVER_PYTHON_OPTS='notebook'
4. Download **Spark-Streaming-Kafka** library from
5. Run the following command

    ```
    >pyspark --master local[2]
    ```

    Result:

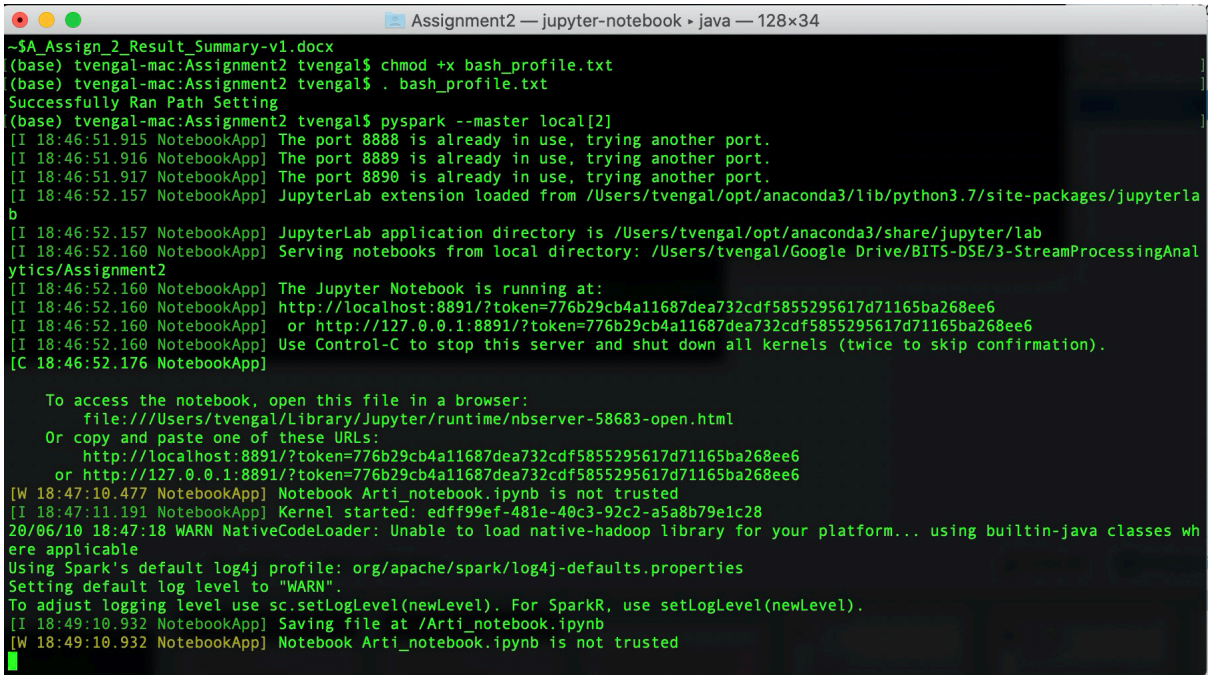


Fig 4: Starting Pyspark

6. This will open up the Jupyter Notebook
7. In a Python Notebook and run the following commands
    a. sc

This will display the SparkContext

b. `os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars {SPARKHOME} /jars/spark-streaming-kafka-0-8_2.11-2.0.2.jar pyspark-shell'`
This will load the spark-streaming-kafka library to PYSPARK and other libraries as posted.

Result:

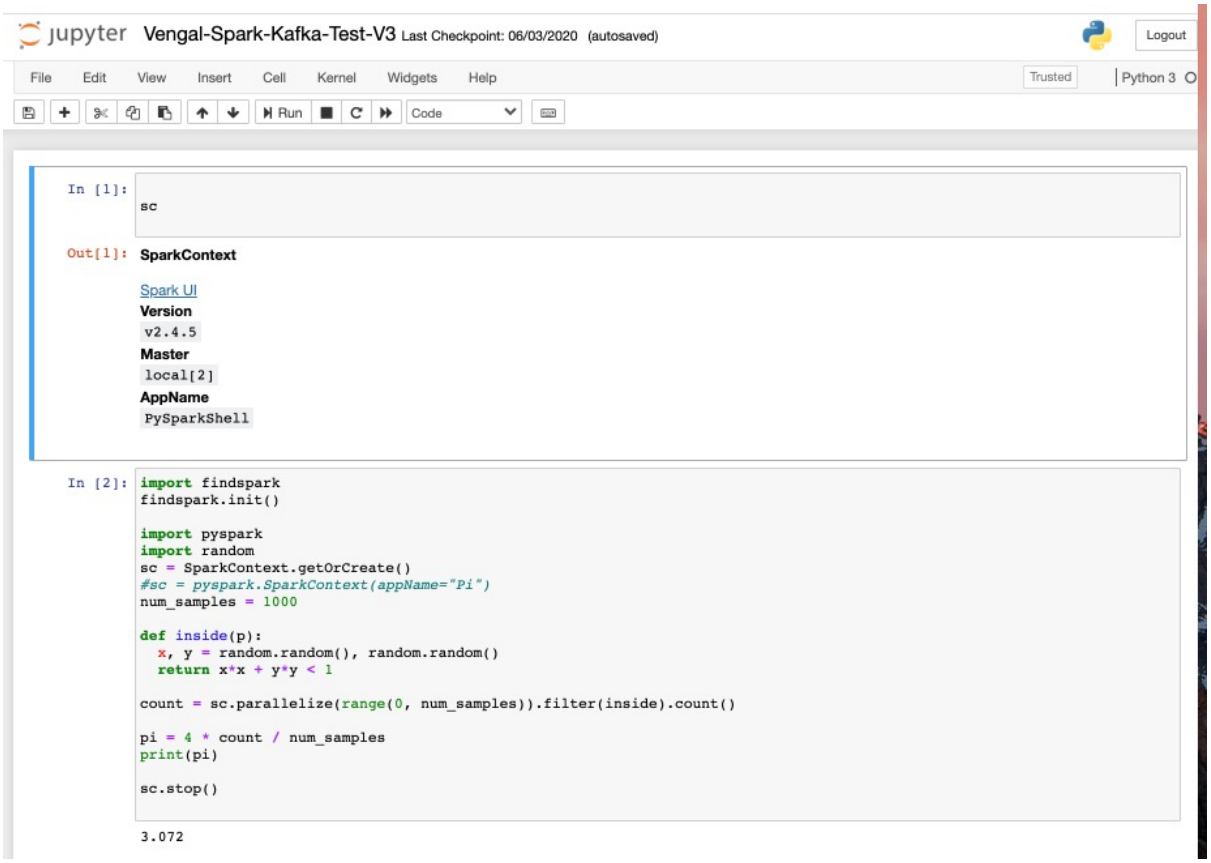


Fig 5: Spark execution on Python Jupyter Notebook

8. Rest of the code is available in the "ReadKafka_RunSpark.ipynb" notebook

Steps and Results for Exercise 1(d): Create a Kafka topic and sending all the data into a topic

Code available at "Send-CSV-to-Kafka-v2.ipynb"

This code has 3 different options to send to Kafka

a) For simple test of sending messages to Kafka in format {"number": #} every second
b) For sending a complete CSV file in JSON without header
c) For sending a complete CSV file in JSON with first line as header

In [1]:
```
from time import sleep
import json
from json import dumps
from kafka import KafkaProducer
import csv
import pandas
import yaml
```

In [2]:
```
producer = KafkaProducer(bootstrap_servers=['localhost:9092'],value_serializer=lambda x: dumps(x).encode('utf-8'))
```

In [5]:
```
#a)for simple test of sending messages to Kafka in format {"number": #} every second

for e in range(100):
    data = {'number' : e}
    producer.send('numtest', value=data)
    sleep(1)
```

In [6]:
```
#b) for sending a complete CSV file in JSON without header
with open('../PS_log-test.csv') as f:
    for line in f:
        row = yaml.safe_load(line)
        jd = json.dumps(row)
        producer.send('csvtopic',jd)  #topic name , data
```

In [8]:
```
#c) for sending a complete CSV file in JSON with first line as header
from kafka import KafkaProducer
import time,csv


# create producer to kafka connection
producer = KafkaProducer(bootstrap_servers='localhost:9092')
# define *.csv file and a char that divide value
fname = "../PS_log-test.csv"
divider_char = ','
# open file
with open(fname) as fp:
    # read header (first line of the input file)
    line = fp.readline()
    header = line.split(divider_char)

    #loop other data rows
    line = fp.readline()
    while line:
        # start to prepare data row to send
        data_to_send = ""
        values = line.split(divider_char)
        len_header = len(header)
        for i in range(len_header):
            data_to_send += "\""+header[i].strip()+"\""+":"+"\""+values[i].strip()+"\""
            if i<len_header-1 :
                data_to_send += ","
        data_to_send = "{"+data_to_send+"}"


        # send data via producer
        producer.send('test', bytes(data_to_send, encoding='utf-8'))
        line = fp.readline()
        # А это так))) на всякий случай
        #time.sleep(1)
producer.close()
```

Fig:6 Jupyter notebook with 3 different options to send Kafka messages

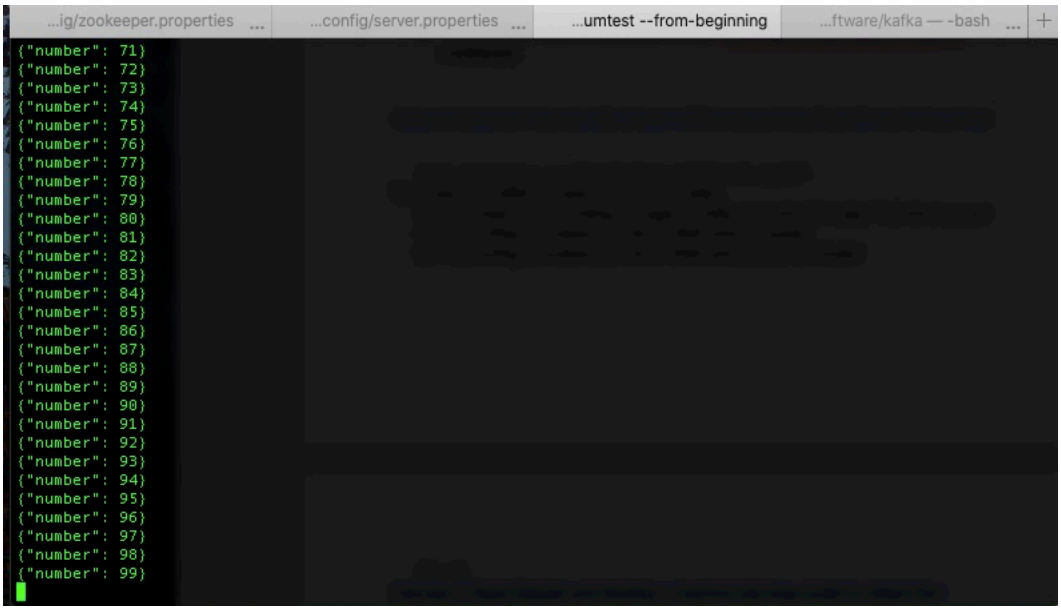


Fig 7(a) Result of sending option (a)

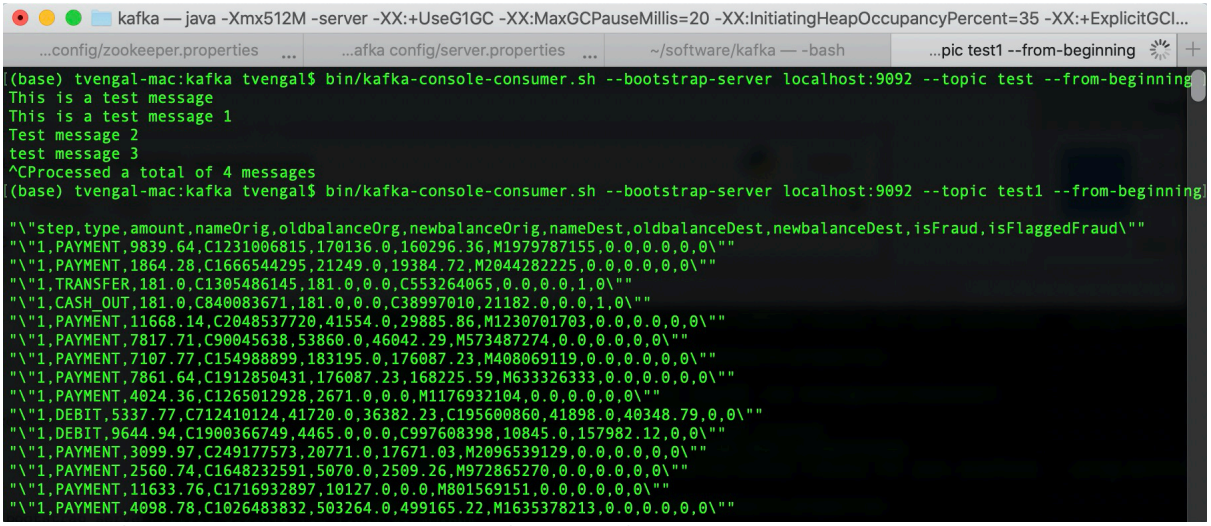


Fig 7(b): Result of sending option (b)

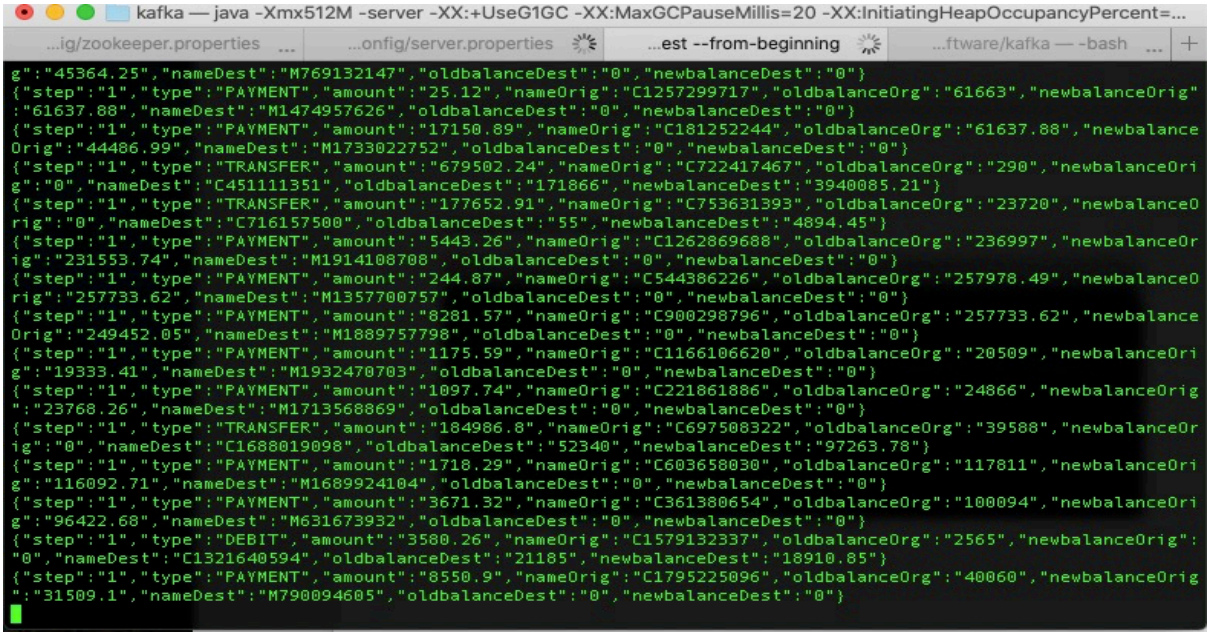


Fig 7(c): Result of sending option (c)

Exercise 2: Read Dataset and develop a machine learning model to detect the customer is good or bad

Scope

- Build a Machine learning model with good accuracy from the given static data.
- This built model should be usable in a stream-processing framework and should be fast enough to detect fraudulent activity in the account.

Summary:
Paysim synthetic dataset of mobile mobile money transactions. Each step represents an hour of simulation. The features present in the dataset are: step, type, amount, nameOrig, nameDest, oldbalanceOrig, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest, isFraud. As part of our analysis we are dropping 'nameOrig'(customer who started the transaction), 'nameDest'(recipient ID of the transaction) because fraudulent transactions are not indicated by these features.

Steps:

1. Import Data: Import dataset to a Python Dataframe
2. Exploratory Data Analysis
    i. List the data, and preview the data and check whether any null values are there

```
In [4]: #List the columns
        list(df)

Out[4]: ['step',
         'type',
         'amount',
         'nameOrig',
         'oldbalanceOrg',
         'newbalanceOrig',
         'nameDest',
         'oldbalanceDest',
         'newbalanceDest',
         'isFraud',
         'isFlaggedFraud']

In [5]: print(df.head())

   step      type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1   PAYMENT   9839.64  C1231006815       170136.0       160296.36
1     1   PAYMENT   1864.28  C1666544295        21249.0        19384.72
2     1  TRANSFER    181.00  C1305486145          181.0            0.00
3     1  CASH_OUT    181.00   C840083671          181.0            0.00
4     1   PAYMENT  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0
```

Test if there any missing values in DataFrame. It turns out there are no obvious missing values but, as we will see below, this does not rule out proxies by a numerical value like 0.

```
In [36]: df.isnull().values.any()

Out[36]: False
```

Fig 8: Reading the data in Python

ii. List the details of the dataset

1. Total number of Records = 6362620
2. Total number of Records with Fraudulent Transactions = 8213
3. Total number of Records with Flagged Fraudulent Transactions = 16
4. Percentage Ratio of Fraudulent Transactions= 0.12908204481801522
5. Total number of Records with type PAYMENT = 2151495
6. Total number of Records with type TRANSFER = 532909

7. Total number of Records with type CASH_IN = 1399284
8. Total number of Records with type CASH_OUT = 2237500
9. Total number of Records with type DEBIT = 41432

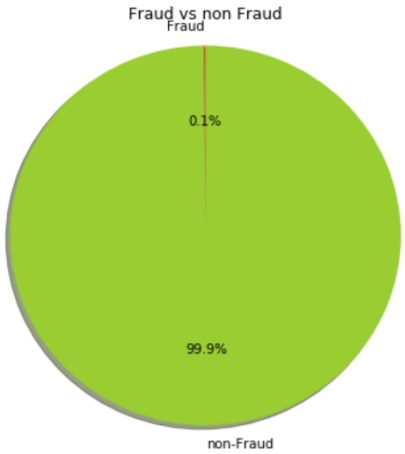


Fig 9 : Pie chart on the Fraud vs Non-Fraud

The feature 'isFlaggedFraud'(flagging a fraudulent transaction (1) and non-fraudulent (0)) is dropped. Because it is not set even in the scenarios of fraudulent transactions.
The target label used here is isFraud which classifies the transactions as fraudulent or not. From our dataset it is observed that 99.9% of the transactions are non-fraudulent and 0.1% are fraudulent as shown below:

iii. Identify the Fraudulent transaction types

- Fraudulent transactions types are ['TRANSFER', 'CASH_OUT']

iv. Count the number of transactions in each of the category

- Total number of Records with fraudulent TRANSFERs = 4097
- Total number of Records with fraudulent CASH_OUTs = 4116
- Total number of Records with isFlaggedfraudulent with TRANSFERs = 16
- Total number of Records with isFlaggedfraudulent with CASH_OUTs = 0

3. Build Machine Learning Model:

We propose 2 classifiers best suitable for the given problem statement and their details are mentioned as below:

i. Two different models were built, evaluated and used
   1. Random Forest based Classifier
   2. XGBoost based Classifier

Following are the details about both the Machine learning Models:

1. Random forest classifier:

The random forest is a classification algorithm consisting of many decision trees. It uses bagging and features randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

In the proposed model, the pre-processing is performed on the dataset where the feature 'type' is converted into categorical data. Also the features like 'nameOrig', 'nameDest' and 'isFlaggedFraud' have been dropped from the dataset. The dataset is split in the ratio of 80:20 such that 80% of the records are used as training data and 20% of the records are used as testing data. The Random forest classifier has been built with n_estimators(number of estimators) as 100 and maximum depth as 10. The evaluation metrics used are 'roc_auc_score' which returned a score of 86% and confusion_matrix.

The trained model is then saved to a pickle file and can be loaded for further can be loaded for further stream processing deployment.

```
Code Available at "FraudPredictionModel-RandomForest.ipynb"
```

2. Extreme gradient boosted (XGBoost) classifier:

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines.Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made.

The pre-processing is performed on the dataset where the feature 'type' is converted into categorical data. Also the features like 'nameOrig', 'nameDest' and 'isFlaggedFraud' have been dropped from the dataset. The dataset is split in the ratio of 80:20 such that 80% of the records are used as training data and 20% of the records are used as testing data. The XGBoost classifier is built with a maximum depth of 3 and n_jobs(number of parallel threads) as 4. The evaluation metric used here is Area under the precision-recall curve (AUPRC) which returned 94.3%.

The trained model is then saved to a binary file and can be loaded for further stream processing deployment.

```
Code Available at "FraudPredictionModel-XGB.ipynb"
```

Exercise 3: Stream Data Processing from Kafka and Sending Alerts to another Kafka

Scope

- Read messages from Kafka
- Pre-process the incoming messages in near real-time
- Import and Run Machine Learning model in near real-time
- Filter out the fraudulent transaction data
- Join the fraudulent transaction data with the original dataset
- Send the fraudulent transaction data to a designated Kafka topic and send the non-fraudulent transaction data to another Kafka topic

Reading messages in Kafka

`Code available at  FraudDetection.py`

a. using Python program
using Streaming Context and Create Stream from *readStream* from Kafka

b. Read data is parsed in json format and stored into another alias

```
from kafka import KafkaProducer
import json

conf = SparkConf().setAppName("FraudDetection").setMaster("local").set("spark.io.compression.codec", "snappy")
sc = SparkContext(conf=conf)
sc.setLogLevel("ERROR")
spark = SparkSession(sc)
sqlContext = SQLContext(sc)

jsonschema = StructType().add("step", StringType()) \
                         .add("type", StringType()) \
                         .add("amount", StringType()) \
                         .add("nameOrig", StringType()) \
                         .add("oldbalanceOrg", StringType()) \
                         .add("newbalanceOrig", StringType()) \
                         .add("nameDest",StringType()) \
                         .add("oldbalanceDest",StringType()) \
                         .add("newbalanceDest",StringType())


df = spark.readStream.format("kafka") \
    .option("kafka.bootstrap.servers","localhost:9092") \
    .option("subscribe", "test") \
    .option("startingOffsets", "earliest").load() \
    .select(from_json(col("value").cast("string"), jsonschema).alias("parsed_mta_values"))
```

Fig 10: Code displaying using Spark Context for reading data from

Pre-processing data after reading from Kafka

1. The standard JSON frame was not used in the next stage a full JSON sending code was used
2. The streaming data was read with a defined Json Schema into a pyspark.sql.dataframe and then converted to the Panda Dataframes for pre-processing

```
print("raw Type", type(raw))
print("raw ", raw)
raw.show()

mta_data = raw.toPandas()
org_mta_data = mta_data
print("mta_data",mta_data)
```

Fig 11: Code for transforming dataframe to Pandas data frame

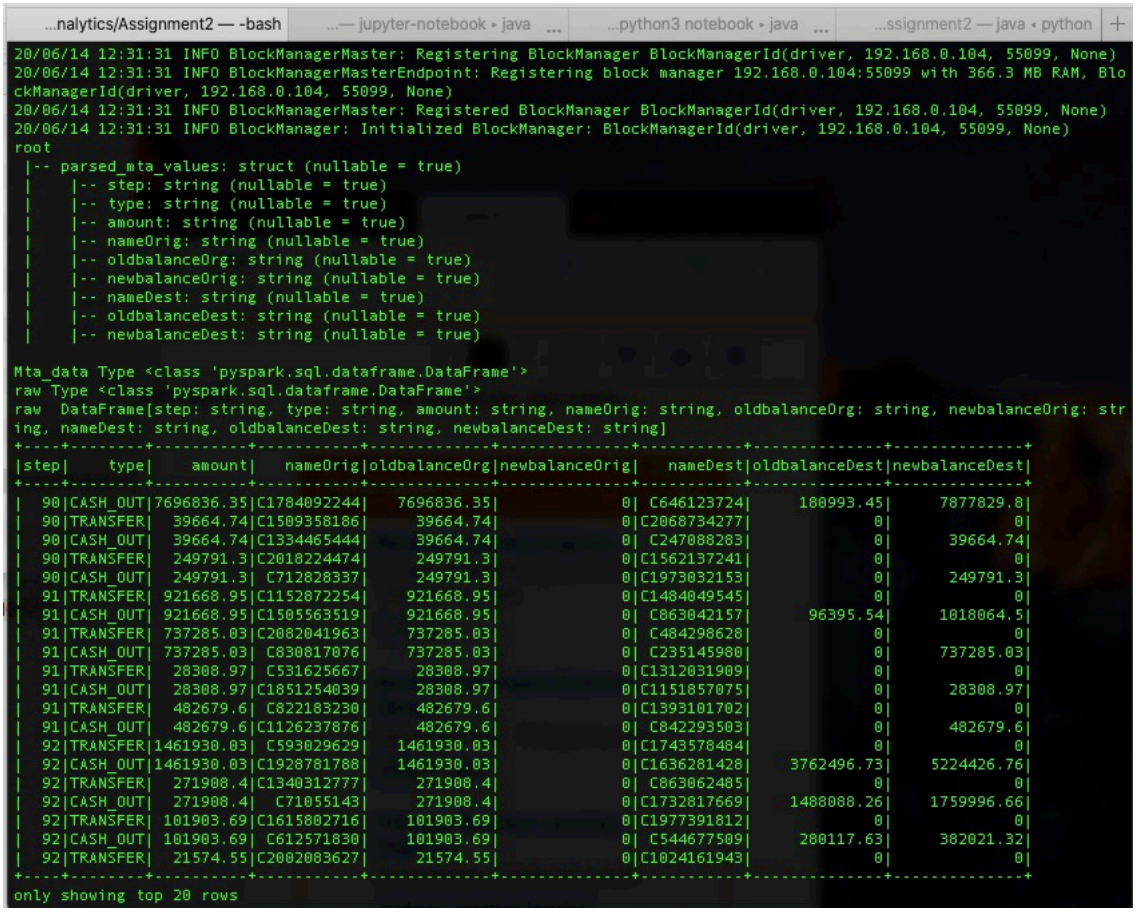
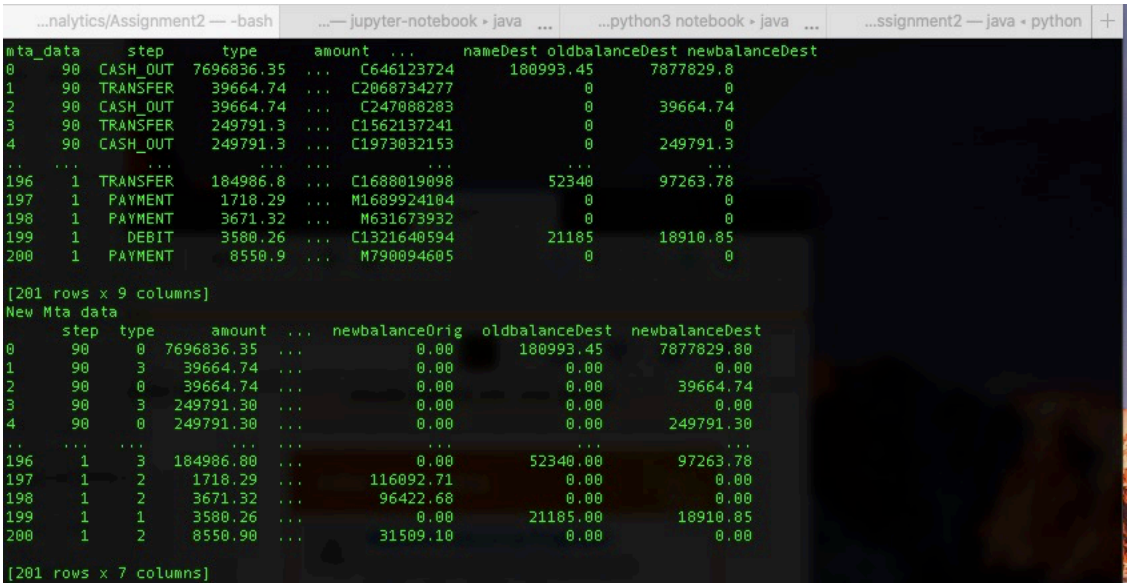


Fig 12: Data received over stream converted into specified Dataframe (pyspark.sql.dataframes type)



```
mta_data     step      type      amount  ...     nameDest oldbalanceDest newbalanceDest
0      90  CASH_OUT  7696836.35  ...   C646123724      180993.45      7877829.8
1      90  TRANSFER    39664.74  ...  C2068734277              0              0
2      90  CASH_OUT    39664.74  ...   C247088283              0       39664.74
3      90  TRANSFER   249791.3   ...  C1562137241              0              0
4      90  CASH_OUT   249791.3   ...  C1973032153              0       249791.3
..    ...       ...         ...  ...          ...            ...            ...
196     1  TRANSFER    184986.8  ...  C1688019098          52340       97263.78
197     1   PAYMENT     1718.29  ...  M1689924104              0              0
198     1   PAYMENT     3671.32  ...   M631673932              0              0
199     1     DEBIT     3580.26  ...  C1321640594          21185       18910.85
200     1   PAYMENT      8550.9  ...   M790094605              0              0

[201 rows x 9 columns]
New Mta data
      step  type      amount  ...  newbalanceOrig  oldbalanceDest  newbalanceDest
0       90     0  7696836.35  ...            0.00       180993.45      7877829.80
1       90     3    39664.74  ...            0.00            0.00            0.00
2       90     0    39664.74  ...            0.00            0.00        39664.74
3       90     3   249791.30  ...            0.00            0.00            0.00
4       90     0   249791.30  ...            0.00            0.00       249791.30
..     ...   ...         ...  ...             ...             ...             ...
196      1     3   184986.80  ...            0.00        52340.00        97263.78
197      1     2     1718.29  ...       116092.71            0.00            0.00
198      1     2     3671.32  ...        96422.68            0.00            0.00
199      1     1     3580.26  ...            0.00        21185.00        18910.85
200      1     2     8550.90  ...        31509.10            0.00            0.00

[201 rows x 7 columns]
```

Fig 13: Data after pre-processing, dropping irrelevant columns and formatting data to numeric type

Note: Tried **reading** as DataFrame object, but then had an error

```
20/06/11 01:56:16 WARN KafkaOffsetReader: Error in attempt 1 getting K
afka offsets:
org.apache.kafka.common.config.ConfigException: Missing required confi
guration "partition.assignment.strategy" which has no default value.
```

Found a solution to use **Kafka 0.10.0.1** client libraries from : https://stackoverflow.com/questions/44959483/pyspark-structured-streaming-kafka-configuration-error?rq=1

Processing Kafka data in Python with Machine Learning Model

- The processed dataframe is passed to Machine Learning Model that was derived from earlier step.
- Classification Result of the ML Model is then added back to the original data
- Only one of the Classifier model is required to be loaded into the Spark Context . We used RandomForest Classifier Model here. (XGB was not used since it required more time and effort, but it can also be used in similar manner)

```
with open('Classifier2.pickle', 'rb') as handle:
        RFmodel = pickle.load(handle)
RFmodel.keys()

RFclf = RFmodel['classifier']
predictions = RFclf.predict(mta_data)
print (predictions)

org_mta_data['isFraud']=predictions
new_mta_data=org_mta_data
print("New Mta data with predictions", new_mta_data)
```

Fig 14: Code for loading Machine Learning Model and sending prepared dataset to Model and joining with original data

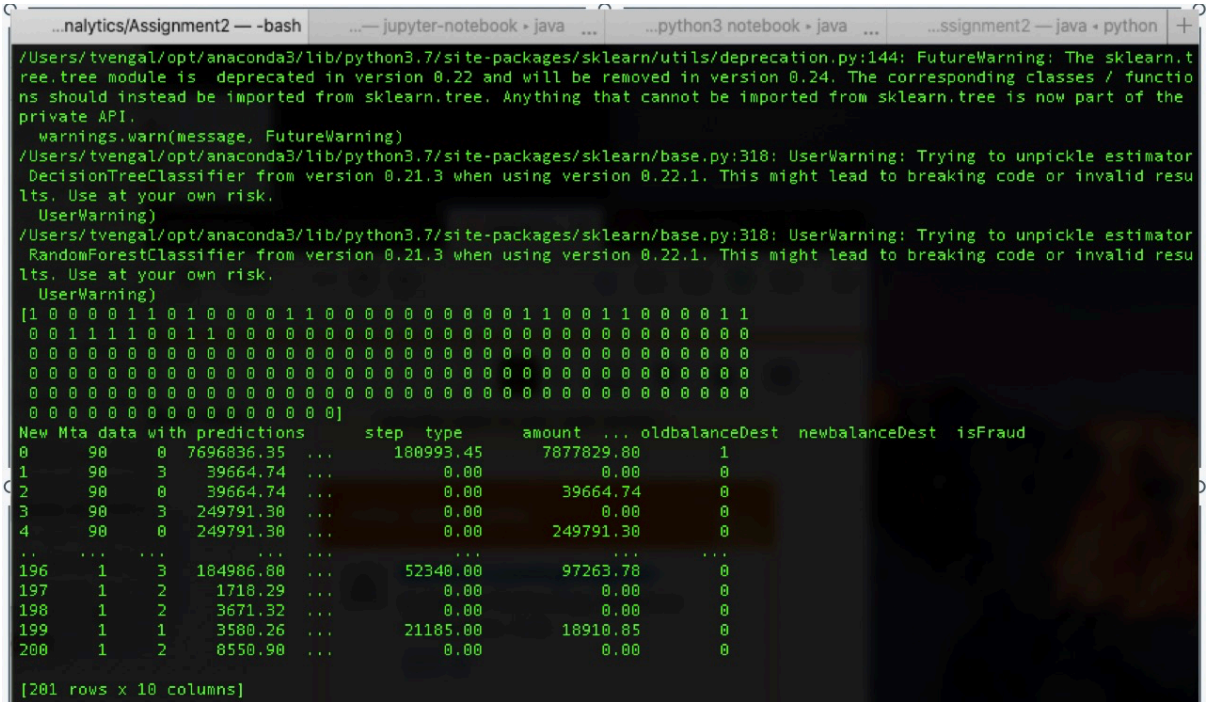


Fig 15: Data result after sending to Machine Learning Model and Merging with the original dataset

For running as a continuous program:

1. Modify the following line

```
raw_table= mta_data.writeStream.outputMode("append").queryName("raw_table").format("memory").start()
```

to

```
raw_table= mta_data.writeStream.outputMode("append").queryName("raw_table").format("memory").start().awaitTermination()
```

2. Run the following command

```
spark-submit --class MainClass --jars /Users/tvengal/opt/anaconda3/lib/python3.7/site-packages/pyspark/jars/kafka-clients-0.10.0.1.jar --driver-class-path /Users/tvengal/opt/anaconda3/lib/python3.7/site-packages/pyspark/jars/kafka-clients-0.10.0.1.jar FraudDetection.py
```

Sending Classified data for Kafka Alert Topic

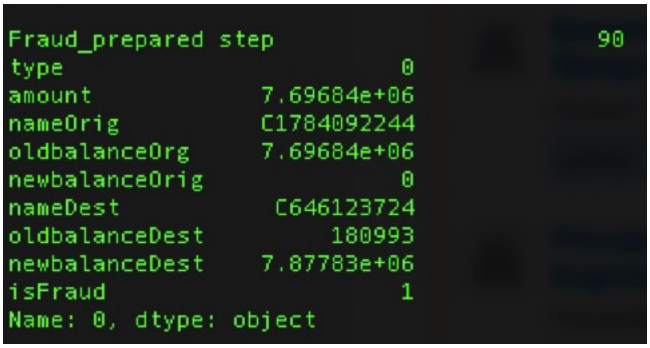


Fig 16: Prepared dataset ready to be sent to Kafka

```
# prepared record => X test which is fraud
for i in range (0,len(new_mta_data)):
        prepared_record=new_mta_data.iloc[i,:]
        if (prepared_record.isFraud==1) :
                print("\nFraud prepared", prepared_record)
                producer = KafkaProducer(bootstrap_servers='localhost:9092',value_serializer=lambda v: json.dumps(v).encode('utf-8'))
                producer.send("FraudData", value=prepared_record.to_json()).get(timeout=30)
        else:
                print("\nNo Fraud prepared", prepared_record)
                producer = KafkaProducer(bootstrap_servers='localhost:9092',value_serializer=lambda v: json.dumps(v).encode('utf-8'))
                producer.send("NonFraudData", value=prepared_record.to_json()).get(timeout=30)
```

Fig 17: Data result after sending to Machine Learning Model and Merging with the original dataset

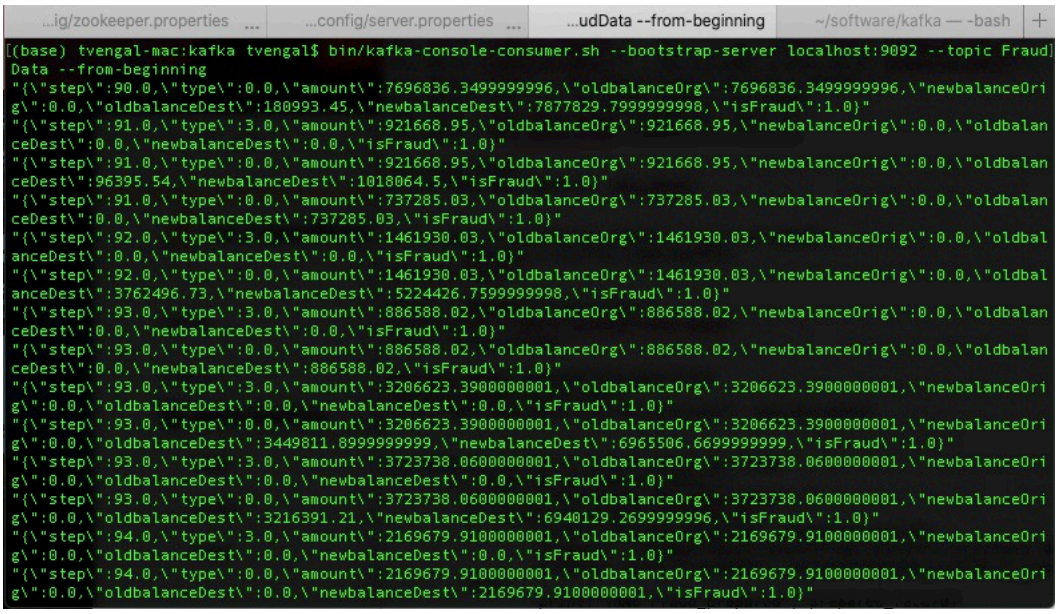


Fig 18 : Fraud Data on a new Kafka topic joined with original data after running through ML Model

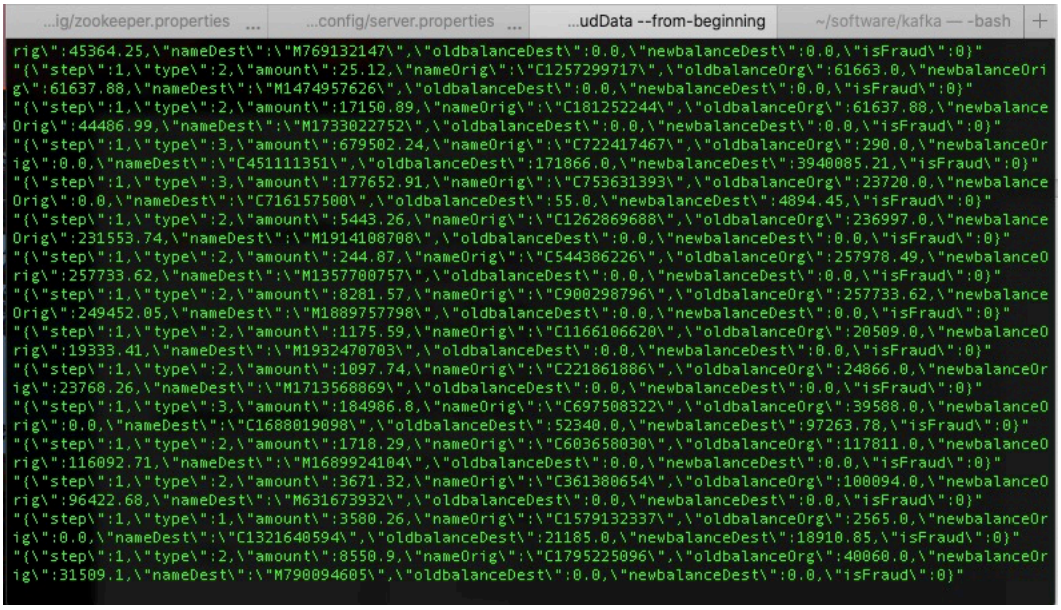


Fig 19: Non-Fraud Data on a new Kafka topic joined with original data after running through ML Model