Capstone Project Final Submission

KafkaSparkStreaming.Java is he driver class for a streaming data processing framework which ingests real-time POS transaction data from Kafka.

NOTE:We already have the details on how data is getting populated in the tables which were created as part of project mid submission.

Below are the steps performed as part of final submission:

<u>Getting the stream of data(JSON format) from Kafka stream</u>:

This was achieved in driver class(KafkaSparkStreaming) using below code [Provided by Upgrad]

```
Logger.getLogger("org").setLevel(Level.OFF);
        Logger.getLogger("akka").setLevel(Level.OFF);
        SparkConf sparkConf = new
SparkConf().setAppName("KafkaSparkStreamingDemo").setMaster("lo
cal");
        JavaStreamingContext jssc = new
JavaStreamingContext(sparkConf, Durations.seconds(1));
        Map<String, Object> kafkaParams = new HashMap<>();
        kafkaParams.put("bootstrap.servers", "100.24.223.181:9092");
        kafkaParams.put("key.deserializer", StringDeserializer.class);
        kafkaParams.put("value.deserializer", StringDeserializer.class);
        kafkaParams.put("group.id", "upgraduserkafkasparknavjot77"); //
change here
        kafkaParams.put("auto.offset.reset", "earliest");
        kafkaParams.put("enable.auto.commit", true);
        Collection<String> topics = Arrays.asList("transactions-topic-
verified");

        JavaInputDStream<ConsumerRecord<String, String>> stream =
KafkaUtils.createDirectStream(jssc,
                LocationStrategies.PreferConsistent(),
                ConsumerStrategies.<String, String>Subscribe(topics,
kafkaParams));
        JavaDStream<String> jds = stream.map(x -> x.value());
```

<u>Cleaning the data:</u> Since we can get some corrupt data in stream there is an additional logic in code to determine if the JSON message is complete or not:

```
      JavaDStream<String> jdsResult = jds.filter(y -> (y!=null &&
y.toString().startsWith("{") && y.toString().endsWith("}")));
```

<u>Connecting to HBase database to get data from Look up table:</u>

This was achieved in class HBaseConnection class where below code was written to connect to HBase DB:

```
org.apache.hadoop.conf.Configuration conf =
(org.apache.hadoop.conf.Configuration) HBaseConfiguration.create();

conf.setInt("timeout", 1200);

conf.set("hbase.master",
"ec2-54-86-225-139.compute-1.amazonaws.com:60000");

conf.set("hbase.zookeeper.quorum",
"ec2-54-86-225-139.compute-1.amazonaws.com");

conf.set("hbase.zookeeper.property.clientPort", "2181");

conf.set("zookeeper.znode.parent", "/hbase");

Connection con = ConnectionFactory.createConnection(conf);

//hbaseAdmin = new HBaseAdmin(conf);
hbaseAdmin = con.getAdmin();
```

Also the EC2 name was updated in hosts/etc file while running the code

Getting the data from HBase DB

Getting the Now once we are able to connect to DB we need to fetch the data for a particular card ID to get the UCL score,Postal Code , Member ID and last transaction date.

NOTE: The table Users_lookup_hb was already created in NoSQL db Hbase(Explaination already provided in Mid Submission)

This was achieved using getStats function where all the fields in the particular record are parsed ,converted to string and stored in variables in class names HBaseDAO

```
Admin hBaseAdmin1 = HbaseConnection.getHbaseAdmin();

        HTable table = null;
        try {
        Get g = new Get(Bytes.toBytes(card_id));
        table = new HTable(hBaseAdmin1.getConfiguration(),
"Users_lookup_hb");

        Result result = table.get(g);

        byte[] uclByte= result.getValue(Bytes.toBytes("cf"),
Bytes.toBytes("ucl"));
        byte[] scoreByte = result.getValue(Bytes.toBytes("cf"),
Bytes.toBytes("score"));
        byte[] postCodeByte= result.getValue(Bytes.toBytes("cf"),
Bytes.toBytes("postcode"));
        byte[] transactionDateByte= result.getValue(Bytes.toBytes("cf"),
Bytes.toBytes("transaction_dt"));

        String ucl = (uclByte != null) ? Bytes.toString(uclByte):"0";
        String score = (scoreByte != null) ?
Bytes.toString(scoreByte):"-1";
        String postCode = (postCodeByte != null) ?
Bytes.toString(postCodeByte):"0";
        String transactionDate = (transactionDateByte != null) ?
Bytes.toString(transactionDateByte):"0";
```

```
    // reading data from look up table
        return new
CardAnalysis(ucl+”,"+score+","+postCode+","+transactionDate);
```

Updating the look up table: This class also had logic to Update the card look up table based on the incoming stream.Function updatePostCodeTranDate was created where postCode and transactionDate were updated:

```
public static boolean updatePostCodeTranDate(String card_id,String
postCode, String transactionDate) throws IOException {

        public static boolean updatePostCodeTranDate(String
card_id,String postCode, String transactionDate) throws IOException
{

        Admin hBaseAdmin1 = HbaseConnection.getHbaseAdmin();

        HTable table = null;
        try {
        Put p = new Put(Bytes.toBytes(card_id));
        table = new HTable(hBaseAdmin1.getConfiguration(),
"Users_lookup_hb");
        p.add(Bytes.toBytes("cf"),
                Bytes.toBytes("postcode"),Bytes.toBytes(postCode));
        p.add(Bytes.toBytes("cf"),

Bytes.toBytes("transaction_dt"),Bytes.toBytes(transactionDate));
        table.put(p);
        return true;

        } catch (Exception e) {

        e.printStackTrace();
        } finally {

        try {

        if (table != null)

        table.close();

        } catch (Exception e) {

        e.printStackTrace();

        }

        }
        return false;

        }
```

Updating the card transactions: This I am assuming that we will be uploading card_transactions.csv to HDFS and load the data into Hive as per part of Mid submission.

Validating if the transaction is fraud or not:

Data Validation: Logic is contained in CardAnalysis.java class. Once we are able to get all the three values(UCL,Score and postal code) , transaction data is then validated based on the three rules' parameters:

1) **Check UCL :** Here we are comparing the current amount to the UCL amount and determining if the current transaction is fraud or not:

```
public boolean checkUCL(double newAmount) {
        if(this.upperControlLimit > 0)
        {
```

```
                    return this.upperControlLimit > newAmount ?
true:false;
            }
            else
                    return true;
    }
```

2) **Check Member Score:** Below is the method used to determine the transaction based on member score:

```
public boolean checkScore() {
            if(this.score > -1)
                    return this.score > 200 ? true:false;
            else
                    return true;
    }
```

3) **Check Postal code:** This method is utilising DistanceUtility to calculate the distance between two postal codes -- This current postal code vs last transaction postal code fetched from the card look up table.Once the distance is calculated we calculate the speed based on the time difference and if the speed is more than 1000 KM/hr we treat that transaction as fraud:

```
public boolean checkPostCode(String newPostCode,String
newTransactionDate) throws ParseException,
NumberFormatException, IOException {
            SimpleDateFormat formatter = new SimpleDateFormat("dd-
MM-yyyy HH:mm:ss");
            DistanceUtility distUtility = new DistanceUtility();
            if(this.transactionDate != "0")
            {
                    double dist =
distUtility.getDistanceViaZipCode(newPostCode, this.postCode);
```

```
                        double time =
(Math.abs(formatter.parse(newTransactionDate).getTime() -
formatter.parse(this.transactionDate).getTime())/ (1000*60*60));
                        // if the speed < 1000 km/hr then GENUNIE else it is
FRAUD
                        return dist/time < 1000 ? true:false;
                }
                else if(newTransactionDate != null)
                        return true;
                else
                        return false;

        }
```

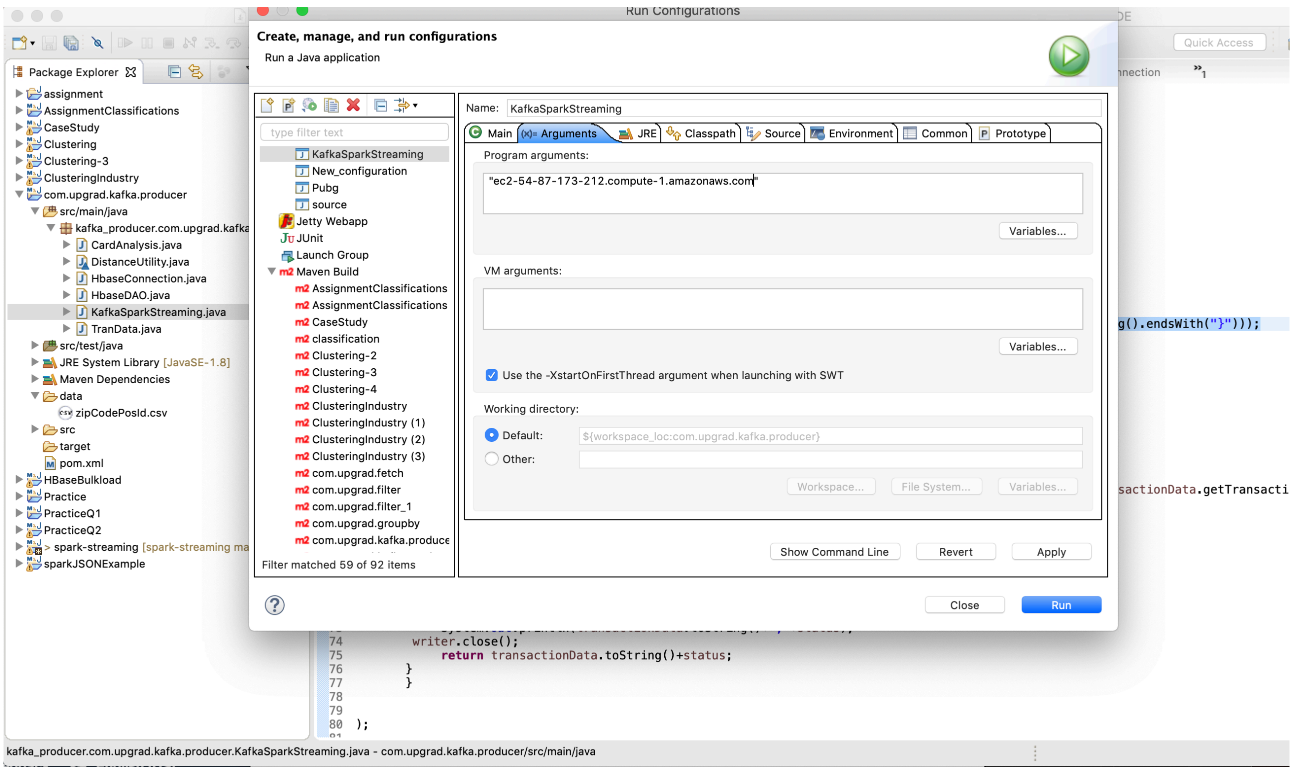
<u>Displaying the output:</u> Once the transaction has been classified as fraud or genuine we display the output along with the driver class in driver class:

```
JavaDStream<String> jdsResult1 = jdsResult.map(new
Function<String,String>(){
           private static final long serialVersionUID = 1L;
           @Override
          public String call(String t) throws IOException,
NumberFormatException, ParseException {
                   ObjectMapper maper = new ObjectMapper();
                   TranData transactionData = maper.readValue(t,
TranData.class);
                  PrintWriter writer = new PrintWriter(new File("data/
test.csv")) ;
               String status=null;
                   CardAnalysis cardStats =
HbaseDAO.getStats(transactionData.getCard_id());
                   if(cardStats != null &&
cardStats.checkUCL(transactionData.getAmount())
                                   && cardStats.checkScore() &&
cardStats.checkPostCode(transactionData.getPostcode(),
transactionData.getTransaction_dt()))
                   {
                        // Transaction validated successfully on all 3 rules
                           status="GENUINE";
```

```
            }
            else
            {
                    // fraud transaction
                    status="FRAUD";
            }
        System.out.println(transactionData.toString()+","+status);

        writer.close();
            return transactionData.toString()+status;
    }
    }
```

Console output screenshot: Please find below the initial screenshot where i have passed my EC2 IP as parameter(Full logs are attached separately)

Argument settings:



Run Configurations

Create, manage, and run configurations

Run a Java application

Name: KafkaSparkStreaming

Main | Arguments | JRE | Classpath | Source | Environment | Common | Prototype

Program arguments:

```
"ec2-54-87-173-212.compute-1.amazonaws.com"
```

VM arguments:

☑ Use the -XstartOnFirstThread argument when launching with SWT

Working directory:

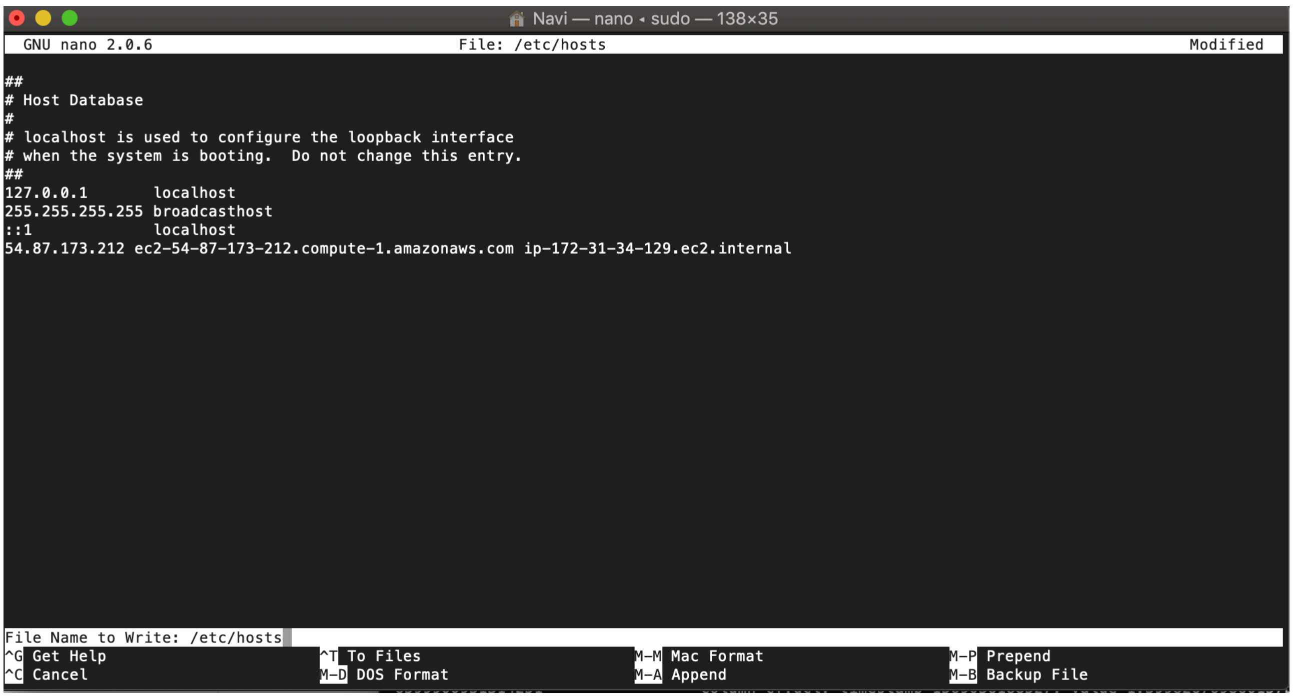
● Default: ${workspace_loc:com.upgrad.kafka.producer}

○ Other:

```
        writer.close();
            return transactionData.toString()+status;
        }
        }


);
```

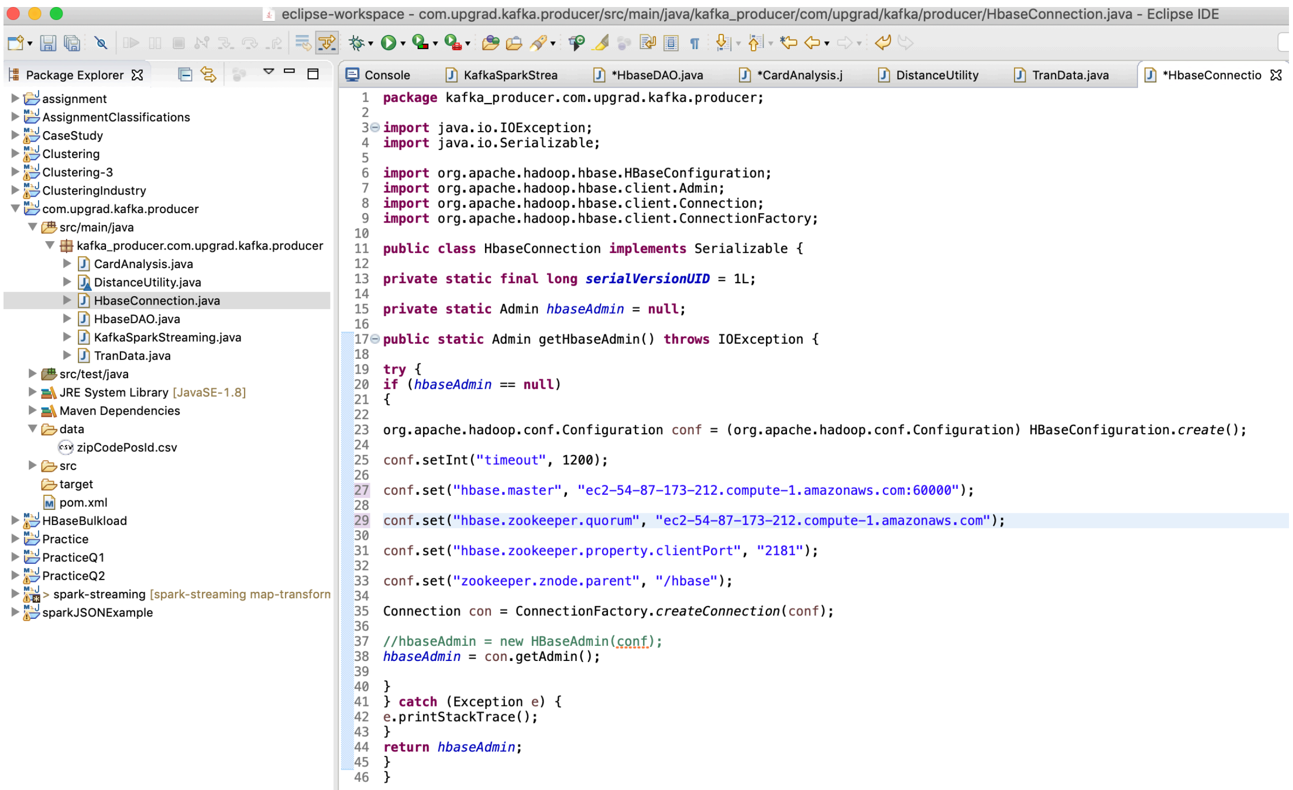
kafka_producer.com.upgrad.kafka.producer.KafkaSparkStreaming.java - com.upgrad.kafka.producer/src/main/java

Etc/host settings:



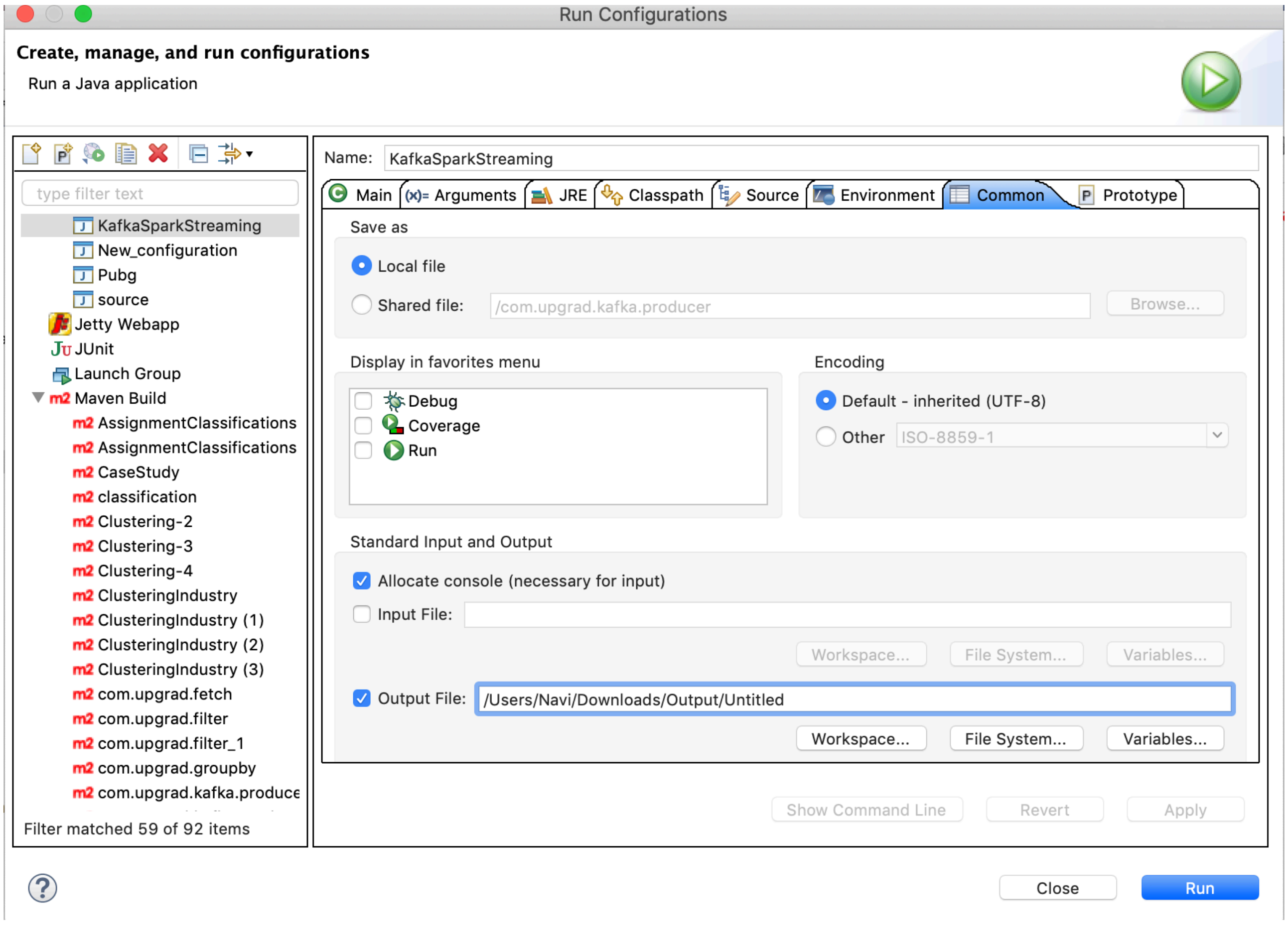
```
  GNU nano 2.0.6                              File: /etc/hosts                                                     Modified

##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
54.87.173.212 ec2-54-87-173-212.compute-1.amazonaws.com ip-172-31-34-129.ec2.internal

File Name to Write: /etc/hosts
^G Get Help                          ^T To Files                          M-M Mac Format                       M-P Prepend
^C Cancel                            M-D DOS Format                       M-A Append                           M-B Backup File
```

HBaseConnection settings:



```java
package kafka_producer.com.upgrad.kafka.producer;

import java.io.IOException;
import java.io.Serializable;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Admin;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;

public class HbaseConnection implements Serializable {

private static final long serialVersionUID = 1L;

private static Admin hbaseAdmin = null;

public static Admin getHbaseAdmin() throws IOException {

try {
if (hbaseAdmin == null)
{

org.apache.hadoop.conf.Configuration conf = (org.apache.hadoop.conf.Configuration) HBaseConfiguration.create();

conf.setInt("timeout", 1200);

conf.set("hbase.master", "ec2-54-87-173-212.compute-1.amazonaws.com:60000");

conf.set("hbase.zookeeper.quorum", "ec2-54-87-173-212.compute-1.amazonaws.com");

conf.set("hbase.zookeeper.property.clientPort", "2181");

conf.set("zookeeper.znode.parent", "/hbase");

Connection con = ConnectionFactory.createConnection(conf);

//hbaseAdmin = new HBaseAdmin(conf);
hbaseAdmin = con.getAdmin();

}
} catch (Exception e) {
e.printStackTrace();
}
return hbaseAdmin;
}
}
```

Saving the console logs in an external file:



Create, manage, and run configurations

Run a Java application

Name: KafkaSparkStreaming

Save as

- [✓ Local file]
- Shared file: /com.upgrad.kafka.producer

Display in favorites menu

- [ ] Debug
- [ ] Coverage
- [ ] Run

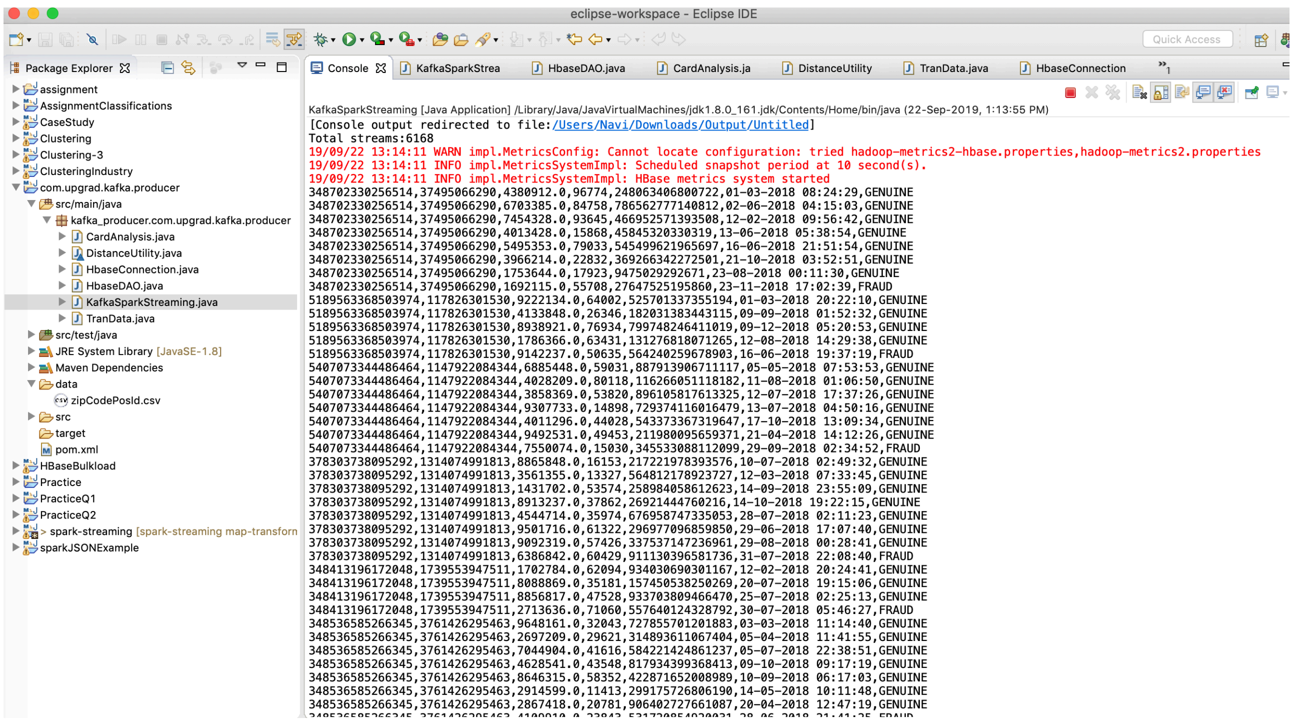
Encoding

- [✓ Default - inherited (UTF-8)]
- Other ISO-8859-1

Standard Input and Output

- [✓] Allocate console (necessary for input)
- [ ] Input File:
- [✓] Output File: /Users/Navi/Downloads/Output/Untitled

Console Output:



```
KafkaSparkStreaming [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java (22-Sep-2019, 1:13:55 PM)
[Console output redirected to file:/Users/Navi/Downloads/Output/Untitled]
Total streams:6168
19/09/22 13:14:11 WARN impl.MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-hbase.properties,hadoop-metrics2.properties
19/09/22 13:14:11 INFO impl.MetricsSystemImpl: Scheduled snapshot period at 10 second(s).
19/09/22 13:14:11 INFO impl.MetricsSystemImpl: HBase metrics system started
348702330256514,37495066290,4380912.0,96774,248063406800722,01-03-2018 08:24:29,GENUINE
348702330256514,37495066290,6703385.0,84758,786562777140812,02-06-2018 04:15:03,GENUINE
348702330256514,37495066290,7454328.0,93645,466952571393508,12-02-2018 09:56:42,GENUINE
348702330256514,37495066290,4013428.0,15868,45845320330319,13-06-2018 05:38:54,GENUINE
348702330256514,37495066290,5495353.0,79033,545499621965697,16-06-2018 21:51:54,GENUINE
348702330256514,37495066290,3966214.0,22832,369266342272501,21-10-2018 03:52:51,GENUINE
348702330256514,37495066290,1753644.0,17923,9475029292671,23-08-2018 00:11:30,GENUINE
348702330256514,37495066290,1692115.0,55708,27647525195860,23-11-2018 17:02:39,FRAUD
5189563368503974,117826301530,9222134.0,64002,525701337355194,01-03-2018 20:22:10,GENUINE
5189563368503974,117826301530,4133848.0,26346,182031383443115,09-09-2018 01:52:32,GENUINE
5189563368503974,117826301530,8938921.0,76934,799748246411019,09-12-2018 05:20:53,GENUINE
5189563368503974,117826301530,1786366.0,63431,131276818071265,12-08-2018 14:29:38,GENUINE
5189563368503974,117826301530,9142237.0,50635,564240259678903,16-06-2018 19:37:19,FRAUD
5407073344486464,1147922084344,6885448.0,59031,887913906711117,05-05-2018 07:53:53,GENUINE
5407073344486464,1147922084344,4028209.0,80118,116266051118182,11-08-2018 01:06:50,GENUINE
5407073344486464,1147922084344,3858369.0,53820,896105817613325,12-07-2018 17:37:26,GENUINE
5407073344486464,1147922084344,9307733.0,14898,729374116016479,13-07-2018 04:50:16,GENUINE
5407073344486464,1147922084344,4011296.0,44028,543373367319647,17-10-2018 13:09:34,GENUINE
5407073344486464,1147922084344,9492531.0,49453,211980095659371,21-04-2018 14:12:26,GENUINE
5407073344486464,1147922084344,7550074.0,15030,345533088112099,29-09-2018 02:34:52,FRAUD
378303738095292,1314074991813,8865848.0,16153,217221978393576,10-07-2018 02:49:32,GENUINE
378303738095292,1314074991813,3561355.0,13327,564812178923727,12-03-2018 07:33:45,GENUINE
378303738095292,1314074991813,1431702.0,53574,258984058612623,14-09-2018 23:55:09,GENUINE
378303738095292,1314074991813,8913237.0,37862,26921444760216,14-10-2018 19:22:15,GENUINE
378303738095292,1314074991813,4544714.0,35974,676958747335053,28-07-2018 02:11:23,GENUINE
378303738095292,1314074991813,9501716.0,61322,296977096859850,29-06-2018 17:07:40,GENUINE
378303738095292,1314074991813,9092319.0,57426,337537147236961,29-08-2018 00:28:41,GENUINE
378303738095292,1314074991813,6386842.0,60429,911130396581736,31-07-2018 22:08:40,FRAUD
348413196172048,1739553947511,1702784.0,62094,934030690301167,12-02-2018 20:24:41,GENUINE
348413196172048,1739553947511,8088869.0,35181,157450538250269,20-07-2018 19:15:06,GENUINE
348413196172048,1739553947511,8856817.0,47528,933703809466470,25-07-2018 02:25:13,GENUINE
348413196172048,1739553947511,2713636.0,71060,557640124328792,30-07-2018 05:46:27,FRAUD
348536585266345,3761426295463,9648161.0,32043,727855701201883,03-03-2018 11:14:40,GENUINE
348536585266345,3761426295463,2697209.0,29621,314893611067404,05-04-2018 11:41:55,GENUINE
348536585266345,3761426295463,7044904.0,41616,584221424861237,05-07-2018 22:38:51,GENUINE
348536585266345,3761426295463,4628541.0,43548,817934399368413,09-10-2018 09:17:19,GENUINE
348536585266345,3761426295463,8646315.0,58352,422871652008989,10-09-2018 06:17:03,GENUINE
348536585266345,3761426295463,2914599.0,11413,299175726806190,14-05-2018 10:11:48,GENUINE
348536585266345,3761426295463,2867418.0,20781,906402727661087,20-04-2018 12:47:19,GENUINE
```