

Real-time Credit card Fraud detection using Spark Streaming, Spark ML, Spark SQL, Kafka, Cassandra and Airflow

1. Introduction	3
1.1 Domain Knowledge	3
1.2 Course description	4
2. Architecture	5
2.1 Spark SQL Job	5
2.2 Spark ML Job	6
2.3 Spark Streaming Job	7
2.4 Fraud-alert dashboard	8
2.5 Airflow Automation	8
3. Image and Code Setup	9
3.1 Clone Code	9
3.2 Import to IntelliJ	11
4. Initial Import Spark Job	16
4.1 Introduction	16
4.2 Transformation	16
4.2.1 Compute Age	16
4.2.2 Compute Distance	17
4.2.3 Save to Cassandra	17
4.3 Demonstration	18
5. Spark Training Job	24
5.1 Introduction	24
5.2 Spark Training Steps	25
5.2.1 Read data from Cassandra	25
5.2.2 String Values to Numeric Values	26
5.2.3 Scaling Numeric Columns	27
5.2.4 Feature Column	30
5.2.5 Pipeline Stages	30
5.2.6 Split data	32
5.2.7 Data Balancing	32
5.2.8 Training	34
5.2.9 Confusion Matrix	35
5.2.10 Save Model	36
5.3 Demonstration	37

6. Spark Streaming	40
6.1 Introduction	40
6.2 Fraud Prediction	41
6.2.1 Read Customer data	41
6.2.2 Load Models	41
6.2.3 Read Offset	41
6.2.4 Fetch Messages from Kafka	41
6.2.5 Prediction	42
6.3 Save Prediction	43
6.4 Save Offset	43
6.5 Exactly-once Semantics	44
6.6 Shutdown Spark Streaming Job	45
6.7 Demonstration	46
7. Airflow Automation	51
7.1 Introduction	51
7.2 Automation Steps	52
7.3 Configure Airflow	54
7.3.1 Create hduser and airflow database	54
7.3.2 Modify airflow.cfg	56
7.4 Build all Projects	58
7.5 Start all the Servers	59
7.6 Airflow Automation	62

Note: In the video course, I am automating both Spark ML and Spark Streaming Job using Apache Airflow. This is just to show how multiple Spark Jobs are automated using Apache Airflow.

In reality, the Spark ML Job will not be automated. It will be manually run once a week, or once a month to create a new model. The model will be tested for efficiency. Also, it will be compared with the efficiency of the previous model. If the new model is better than the previous model then the new model will be deployed. Spark Streaming Job will be restarted to pick the new model. Hence only Spark Streaming Job must be automated in a real scenario

1. Introduction



<https://pixipanda.com/>

Refer to this website for complete training content on Kafka, Spark, Spark SQL, Spark Streaming, Structured Streaming, Hive, Sqoop, HDFS, and MapReduce. Along with training content, I have also uploaded code and image for practice.

1.1 Domain Knowledge

Credit card fraud detection Domain Knowledge

Let's say you own a credit card. Your prior spending habits will be learned. Like how much amount you spend, at which merchant you spend, at what frequency you spend, what do you purchase, etc.

Based on this learning, your current credit transaction will be predicted as fraud if it deviates from your earlier spending habits otherwise it will be treated as a genuine transaction. And fraud transactions will be alerted in the dashboard.

Such predictions will be done on millions of transactions. Hence distributed frameworks are used which can scale as the transactions increase.

1.2 Course Description

Real-time Credit card Fraud Detection is implemented using Spark Kafka and Cassandra.

Spark ML Pipeline Stages like String Indexer, Vector Slicer, Standard Scaler and Vector Assembler is used for Preprocessing

Machine Learning model is created using the Random Forest Algorithm

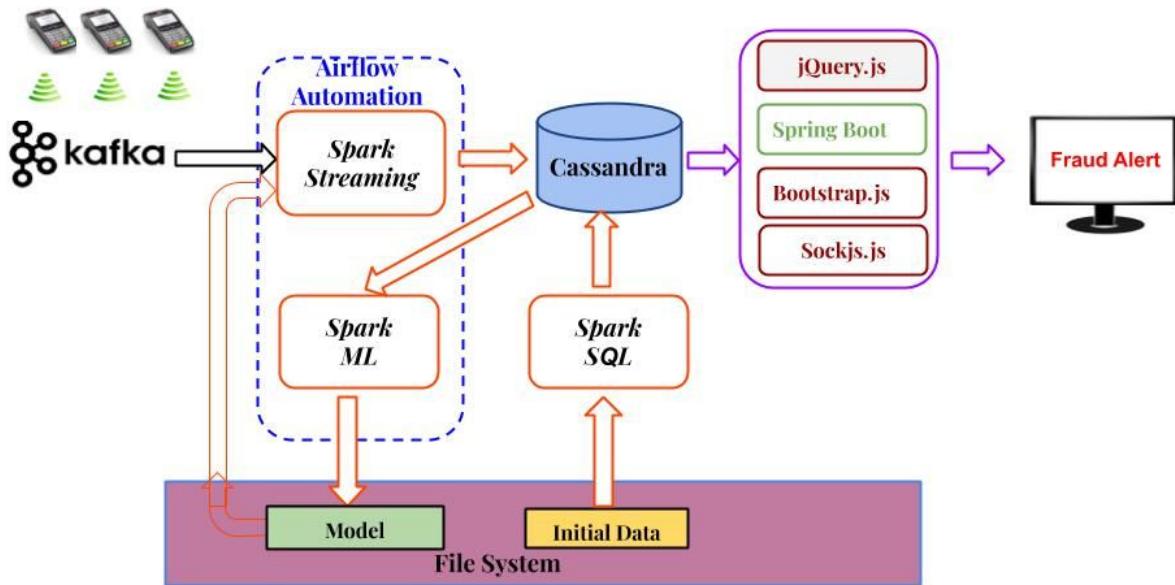
Data balancing is done using K-means Algorithm

Integration of Spark Streaming with Kafka and Cassandra

Exactly-once semantics is achieved using Spark Streaming custom offset management

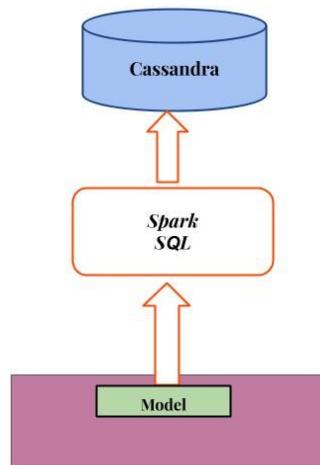
Airflow Automation framework is used to automate Spark Jobs on Spark Standalone Cluster.

2. Architecture



All the transaction data, ie historic data, captured from all the customers are stored in a File System. This file system could be HDFS or Amazon S3 or Azure Blob or it could be any other File System

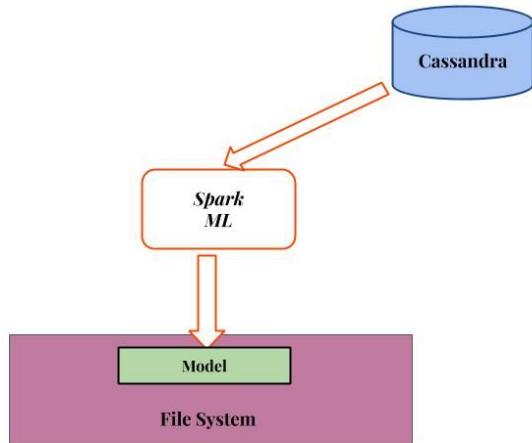
2.1 Spark SQL Job



This Spark Job will import all the transaction data from the file system into the Cassandra database.

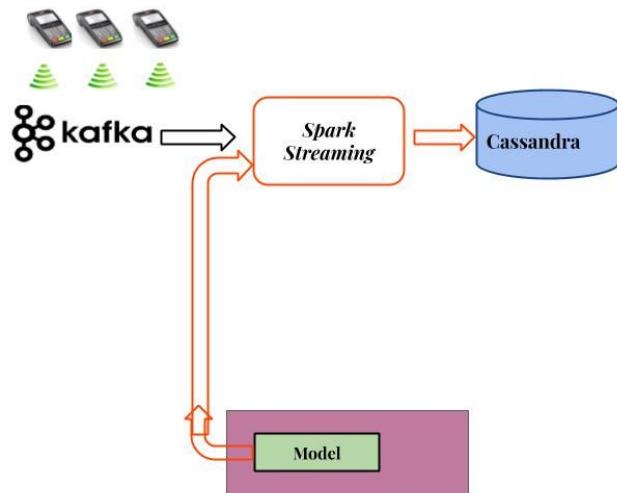
This job will import fraud transactions to fraud table and non-fraud transactions to a non-fraud table

2.2 Spark ML Job



This Spark Job will read fraud and non-fraud transactions from Cassandra. It will train on this data and it will create a model. This model will be saved to the file system.

2.3 Spark Streaming Job

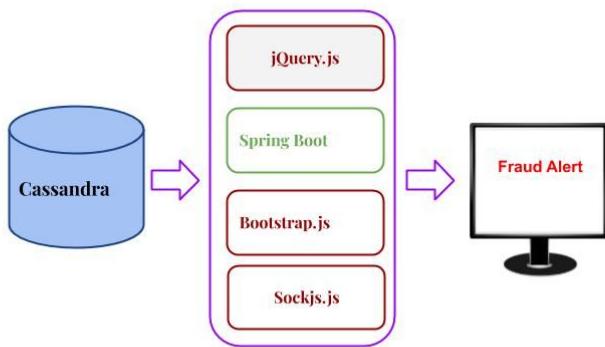


Streaming Job will load the model from the file system. It will then start consuming credit card transaction messages from Kafka.

Using this model it will predict whether a transaction is a fraud or not.

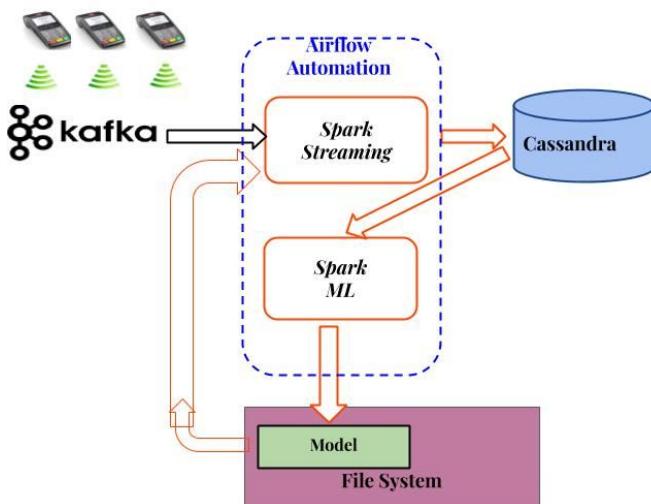
Finally, it will save the predictions to the Cassandra database. Fraud transactions are saved to the fraud table and the non-fraud transactions to the non-fraud table.

2.4 Fraud-alert dashboard



Fraud alert dashboard will internally query Cassandra for the lastest fraud transactions. If there are any fraud transactions, it will be displayed on the dashboard.

2.5 Airflow Automation



Airflow Automation is used to automate Spark Streaming and Spark ML Job. Here we are automating Spark ML Job also. But in reality, Spark ML Job will be run manually by DataScience Team. It will be manually run once a week, or once a month to create a new model. The new model will be tested for efficiency. Also, it will be

compared with the efficiency of the previous model. If the new model is better than the previous model then the new model will be deployed.

Currently running Spark Streaming Job will be stopped and a new Streaming Job will be started to pick the new Model. Airflow automation is used to Stop and Start a new Spark Streaming Job whenever a new model is created.

3. Image and Code Setup

Download and Install Oracle VirtualBox

<https://www.virtualbox.org/wiki/Downloads>

Import Ubuntu image.

http://bit.ly/udemy_ubuntu_image

Username: hduser

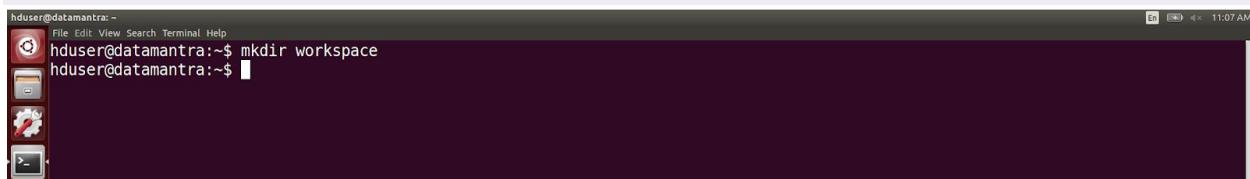
Password: hadoop123

3.1 Clone Code

Create a workspace directory

Create a workspace directory in your home directory

```
mkdir workspace  
cd workspace
```



hduser@datamantra:~\$ mkdir workspace
hduser@datamantra:~\$



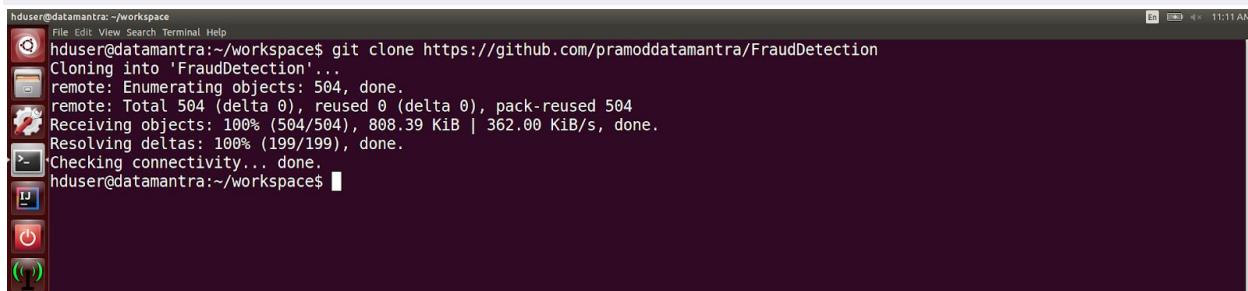
hduser@datamantra:~/workspace\$ cd workspace/
hduser@datamantra:~/workspace\$

Clone FraudDetection from GitHub

This is a branch code. It is a bit different from the code shown in the video.

Here we have VectorSlicer, StandardScaler and Confusion matrix.

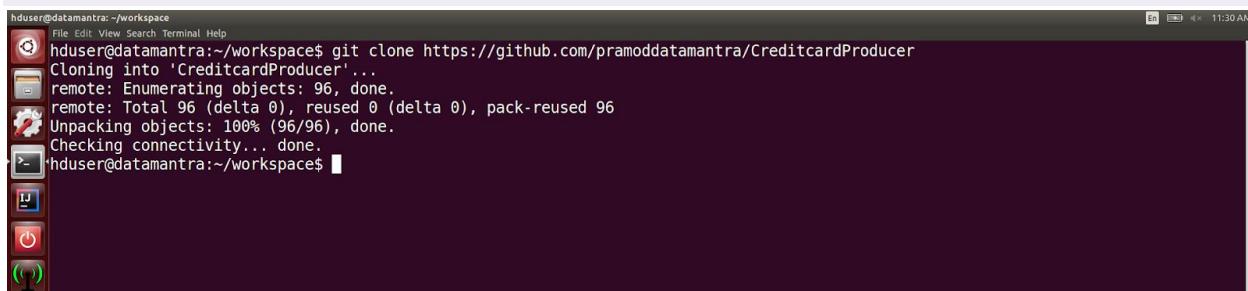
```
git clone  
https://github.com/pramoddatamantra/FraudDetection/tree/FraudDetection\_n\_ML
```



A terminal window titled "Terminal" with the command "git clone https://github.com/pramoddatamantra/FraudDetection" entered. The output shows the cloning process: "Cloning into 'FraudDetection'...", "remote: Enumerating objects: 504, done.", "remote: Total 504 (delta 0), reused 0 (delta 0), pack-reused 504", "Receiving objects: 100% (504/504), 808.39 KiB | 362.00 KiB/s, done.", "Resolving deltas: 100% (199/199), done.", and "Checking connectivity... done." The terminal is located in a workspace named "datamantra".

Clone Credit card producer from GitHub

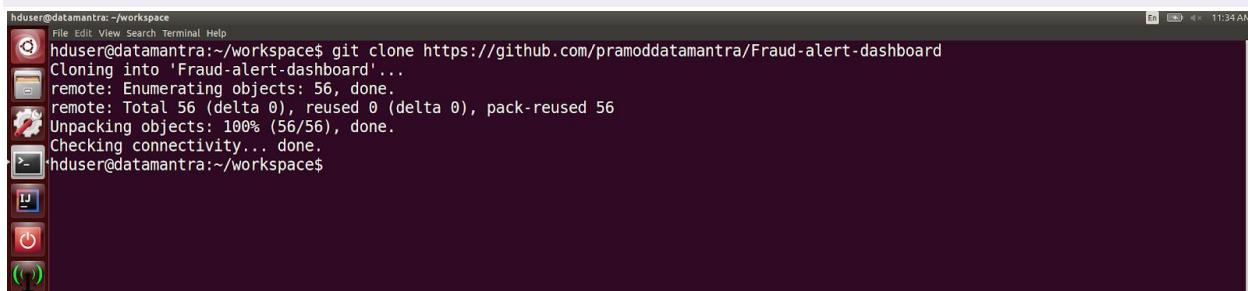
```
git clone https://github.com/pramoddatamantra/CreditcardProducer
```



A terminal window titled "Terminal" with the command "git clone https://github.com/pramoddatamantra/CreditcardProducer" entered. The output shows the cloning process: "Cloning into 'CreditcardProducer'...", "remote: Enumerating objects: 96, done.", "remote: Total 96 (delta 0), reused 0 (delta 0), pack-reused 96", "Unpacking objects: 100% (96/96), done.", and "Checking connectivity... done." The terminal is located in a workspace named "datamantra".

Clone Fraud-alert-dashboard

```
git clone https://github.com/pramoddatamantra/Fraud-alert-dashboard
```

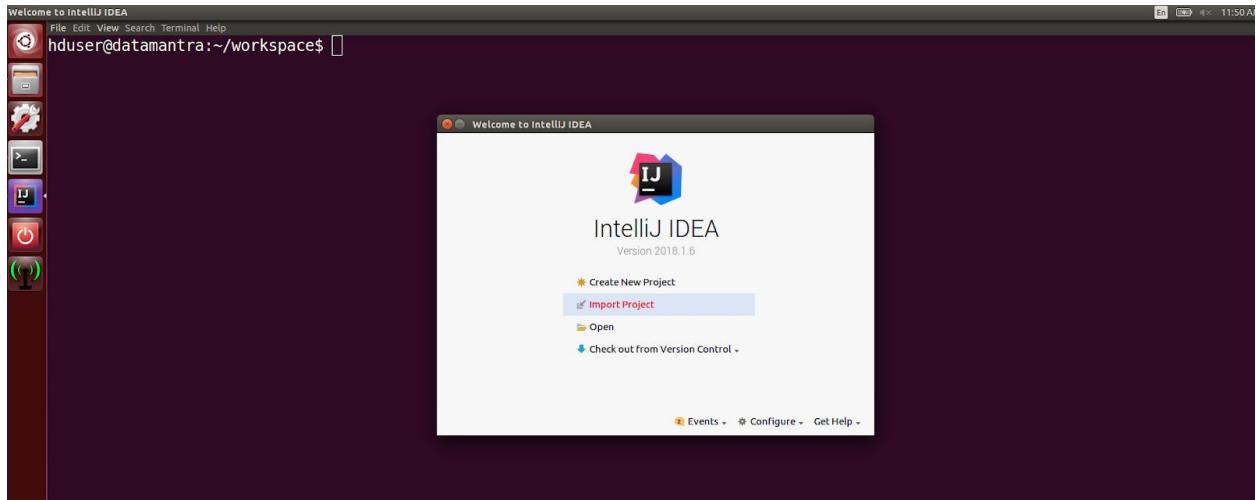


A terminal window titled "Terminal" with the command "git clone https://github.com/pramoddatamantra/Fraud-alert-dashboard" entered. The output shows the cloning process: "Cloning into 'Fraud-alert-dashboard'...", "remote: Enumerating objects: 56, done.", "remote: Total 56 (delta 0), reused 0 (delta 0), pack-reused 56", "Unpacking objects: 100% (56/56), done.", and "Checking connectivity... done." The terminal is located in a workspace named "datamantra".

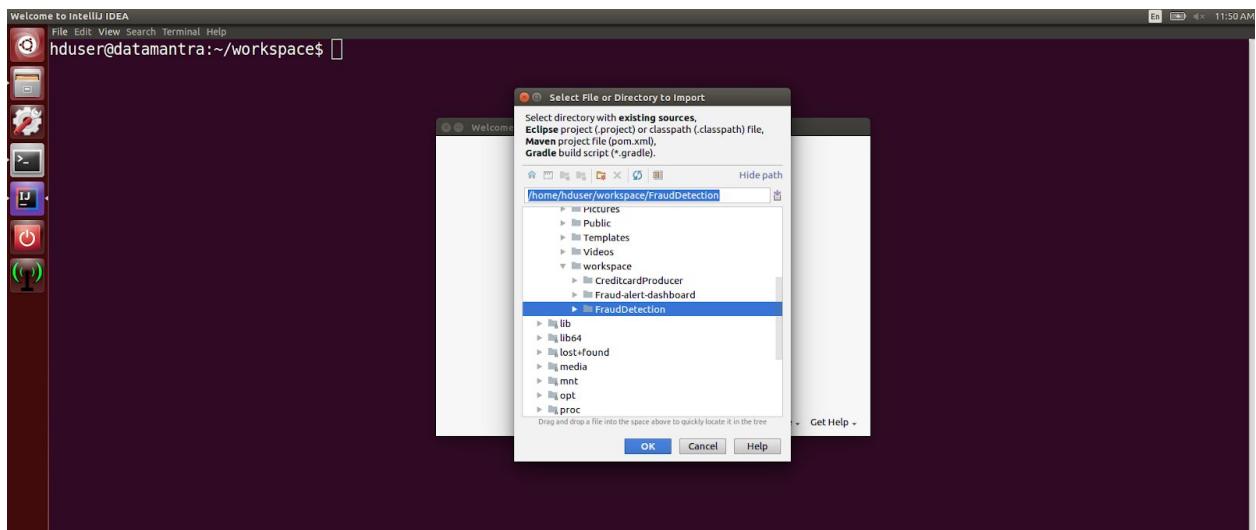
3.2 Import to IntelliJ

Import FraudDetection to IntelliJ

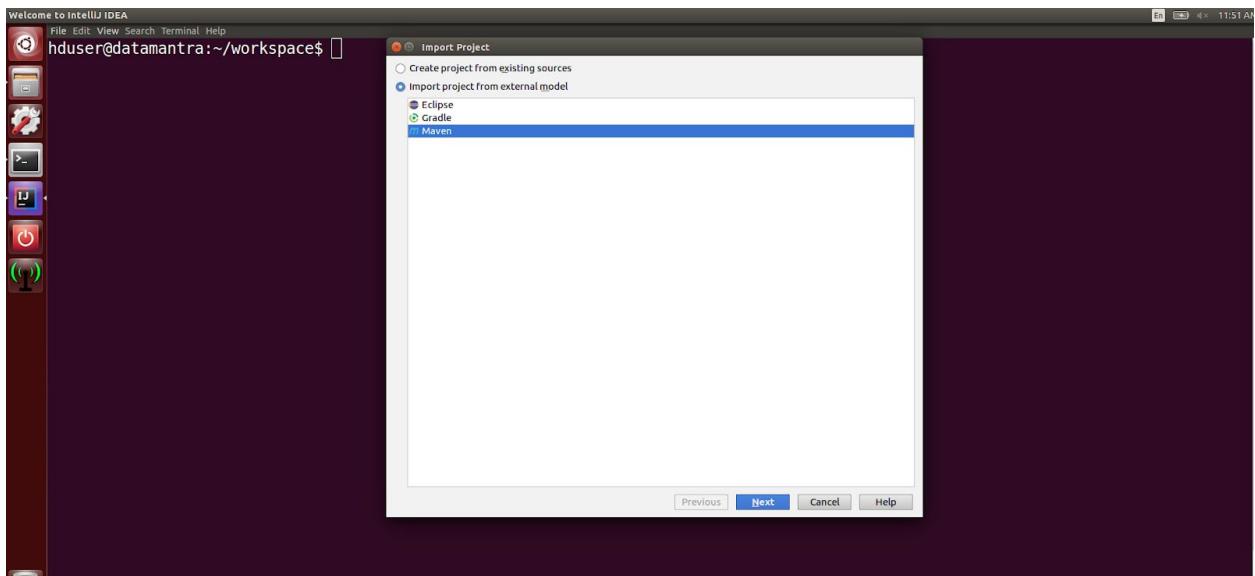
Click Import



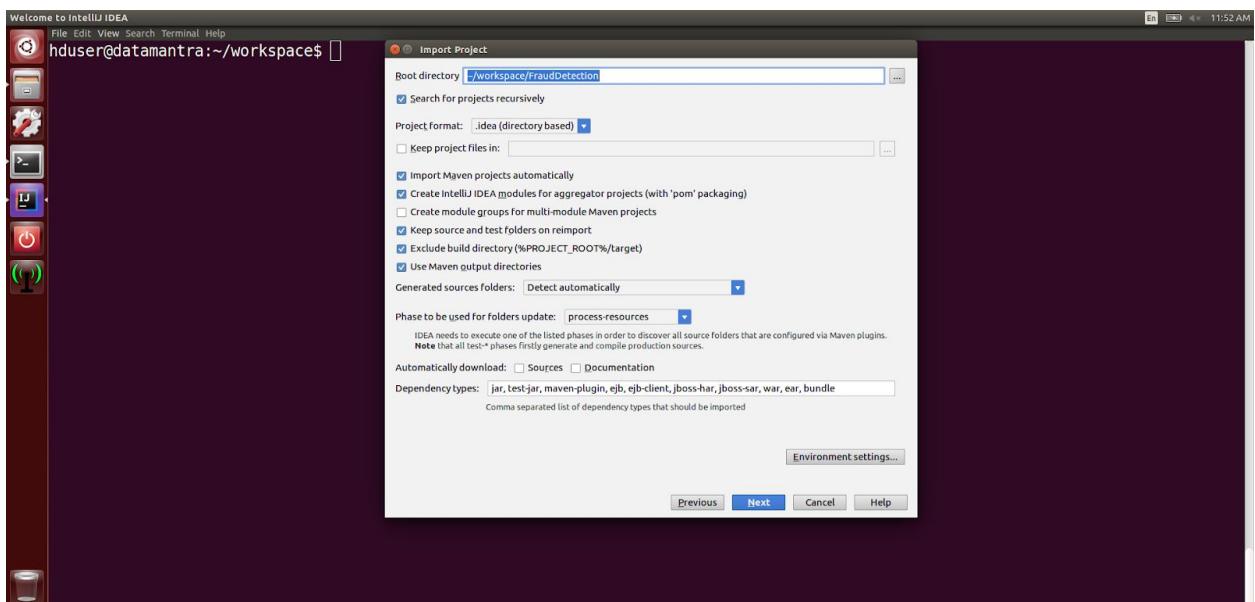
Select FraudDetection Project. Click OK. Click Next



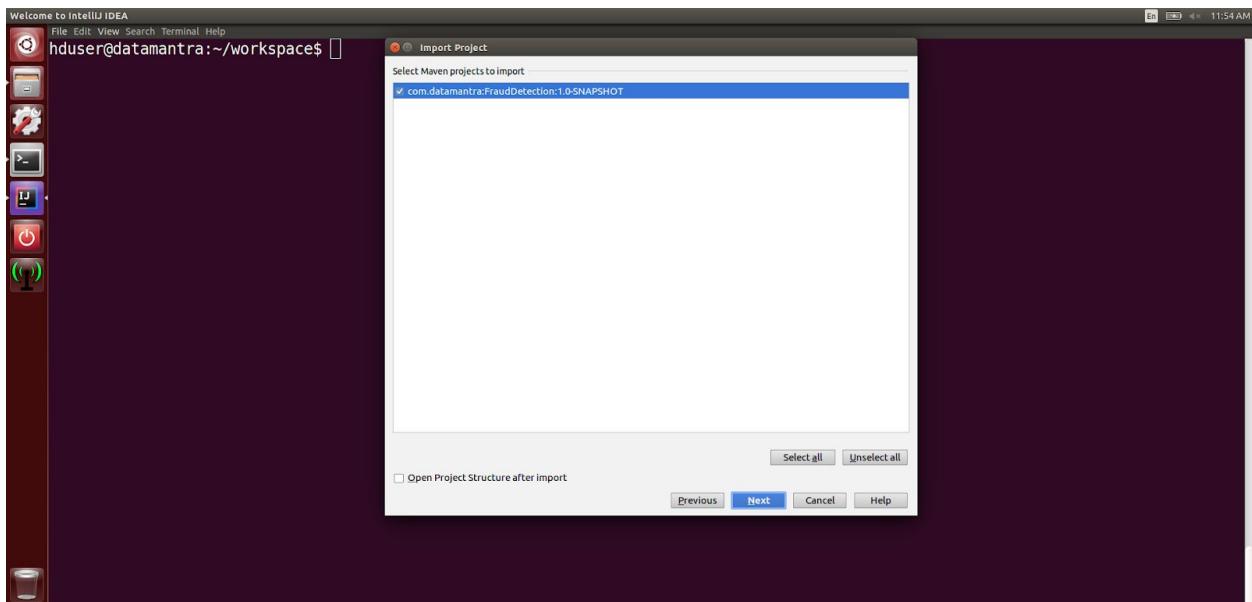
Select Maven. Click Next



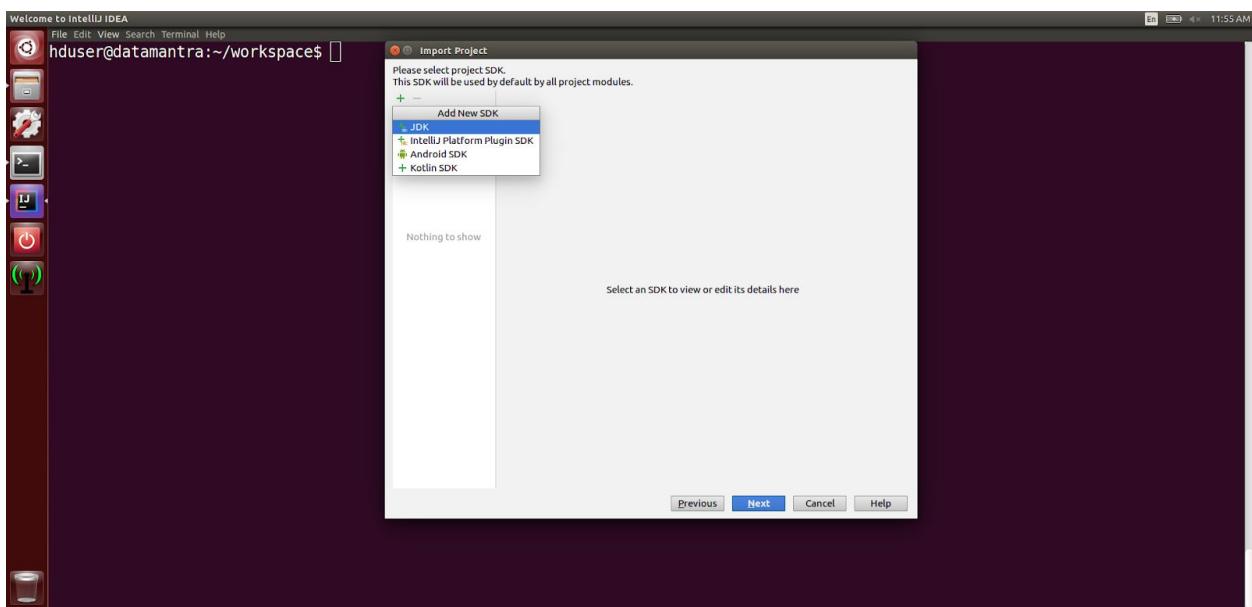
Checkmark “Search for projects recursively” and “Import Maven projects automatically”
Click Next



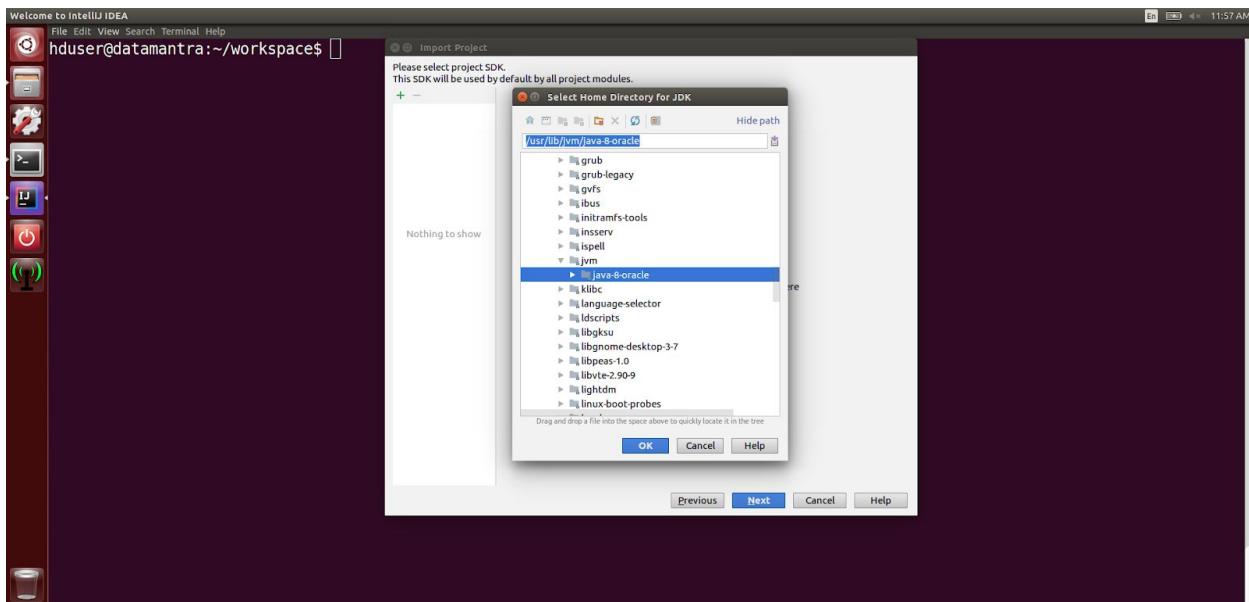
By default, the FraudDetection project will be selected. Click Next



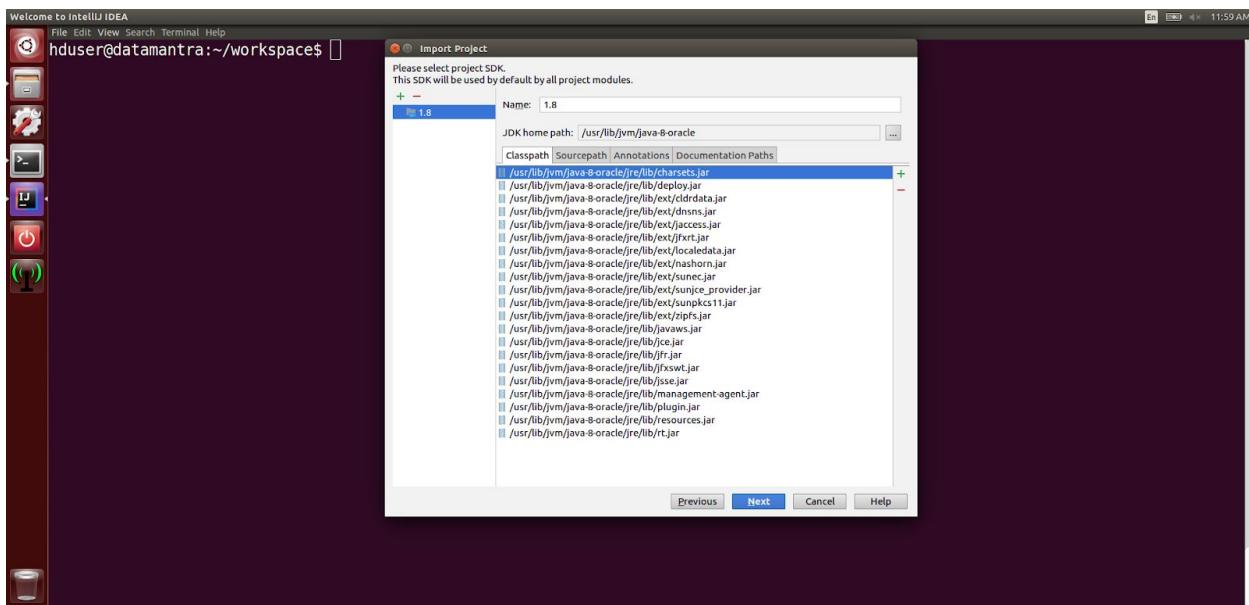
Select JDK



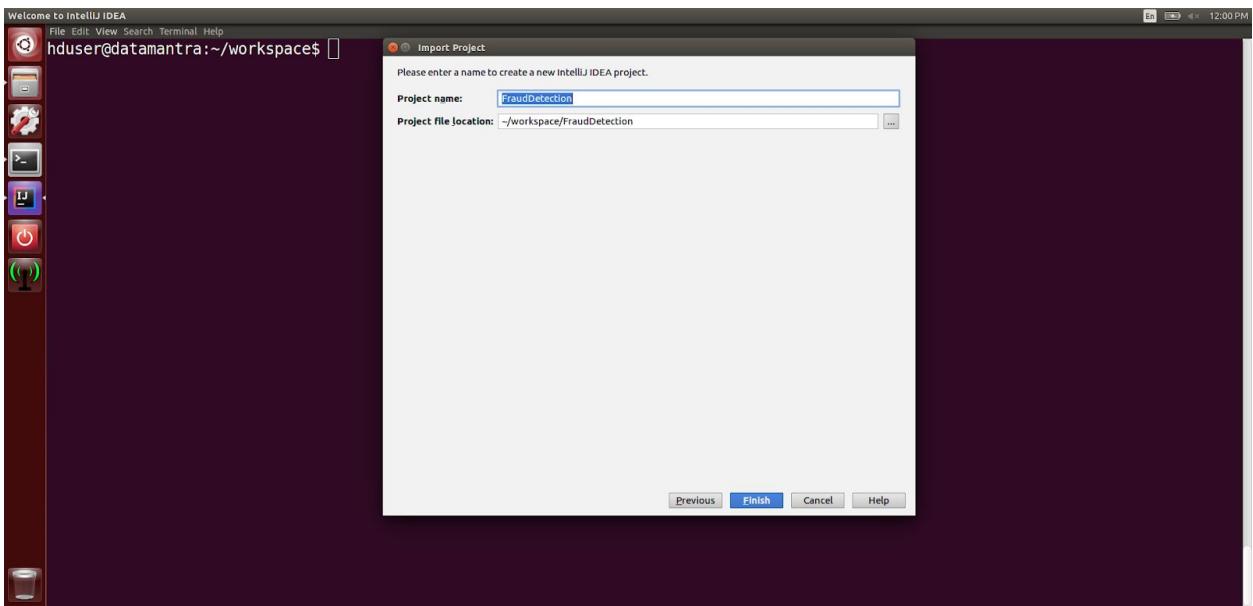
Select the java-8-oracle directory. Click OK and Click Next



Java 1.8 SDK is selected. Click Next



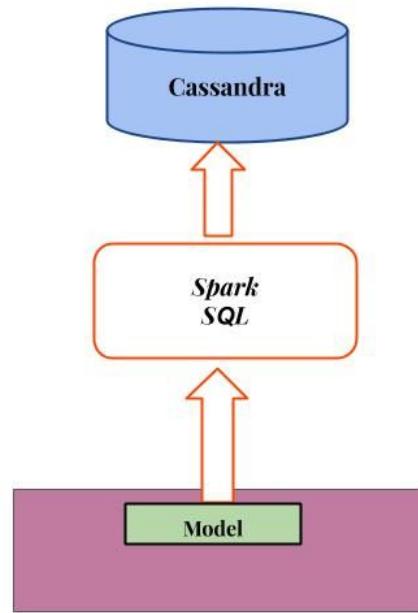
Let the project name and project path be as it is. Click Finish



Follow the above steps to import CreditcardProducer and Fraud-alert-dashboard project into IntelliJ

4. Initial Import Spark Job

4.1 Introduction



This Spark Job will read the customer data and transaction data from the filesystem. It will do some ETL transformations on customer and transaction data, like computing age of the customer, distance between customer and merchant place, etc. It will save these transformed into Cassandra. It will save the customers data to customer table Fraud transactions will be saved to fraud_transaction table and non-fraud transactions to non_fraud_transaction.

4.2 Transformation

4.2.1 Compute Age

Customer.csv has a column with DOB of the customer. From DOB we will age of the customer. Age will be used as a feature in the Spark ML Job

```
val customerDF = DataReader.read(SparkConfig.customerDatasource,  
Schema.customerSchema)
```

```
val customerAgeDF = customerDF.withColumn("age",
(datediff(current_date(),to_date($"dob"))/365).cast(IntegerType))
```

4.2.2 Compute Distance

Customer.csv has customer's home location(latitude and longitude). Transactions.csv has Merchant Location(latitude and longitude). Using this information we will compute the distance between Customer and Merchant. Again distance will be used as a feature in Spark ML Job

```
val transactionDF = DataReader.read(SparkConfig.transactionDatasouce,
Schema.fraudCheckedTransactionSchema)
    .withColumn("trans_date", split($"trans_date", "T").getItem(0))
    .withColumn("trans_time", concat_ws(" ", $"trans_date",
"trans_time"))
    .withColumn("trans_time", to_timestamp($"trans_time", "YYYY-MM-dd
HH:mm:ss") cast(TimestampType))

val distanceUdf = udf(Utils.getDistance _)

val processedDF = transactionDF.join(broadcast(customerAgeDF),
Seq("cc_num"))
    .withColumn("distance", lit(round(distanceUdf($"lat", $"long",
$"merch_lat", $"merch_long"), 2)))
    .select("cc_num", "trans_num", "trans_time", "category", "merchant",
"amt", "merch_lat", "merch_long", "distance", "age", "is_fraud")
```

4.2.3 Save to Cassandra

Customer data is saved to the customer table

Fraud transactions data is saved to fraud_transaction table

Genuine transactions data is saved to non_fraud_transaction table

```
/* Save Customer data to Cassandra */
customerDF.write
    .format("org.apache.spark.sql.cassandra")
    .mode("append")
    .options(Map("keyspace" -> CassandraConfig.keyspace, "table" ->
CassandraConfig.customer))
    .save()
```

```

/* Save fraud transaction data to fraud_transaction cassandra table*/
fraudDF.write
    .format("org.apache.spark.sql.cassandra")
    .mode("append")
    .options(Map("keyspace" -> CassandraConfig.keyspace, "table" ->
CassandraConfig.fraudTransactionTable))
    .save()

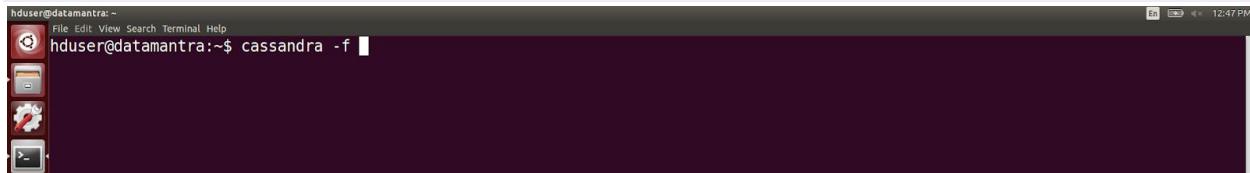
/* Save non fraud transaction data to non_fraud_transaction cassandra
table*/
nonFraudDF.write
    .format("org.apache.spark.sql.cassandra")
    .mode("append")
    .options(Map("keyspace" -> CassandraConfig.keyspace, "table" ->
CassandraConfig.nonFraudTransactionTable))
    .save()

```

4.3 Demonstration

Start Cassandra Server

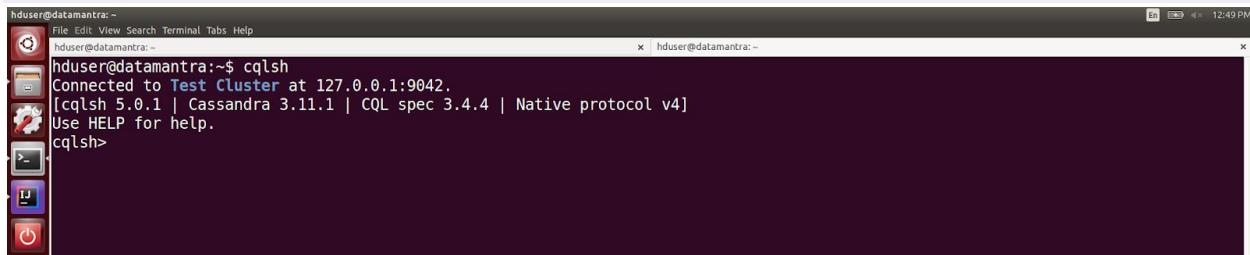
Cassandra -f



Connect to Cassandra Server

Connect to Cassandra Server using Cassandra Cqlsh client

cqlsh



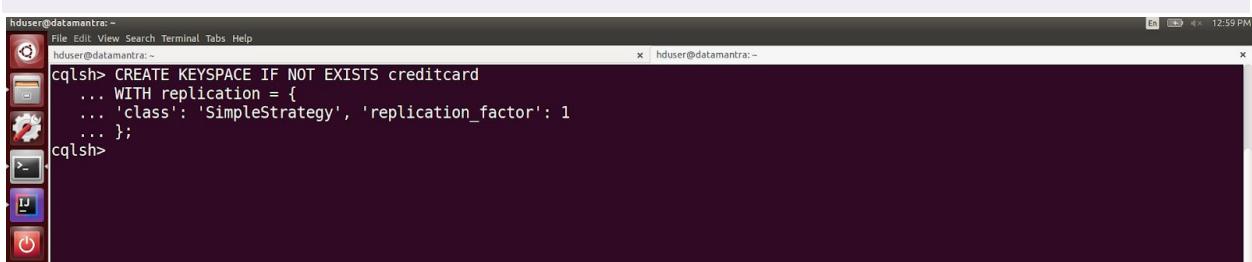
Create keyspace and tables

<https://github.com/pramoddattamantra/FraudDetection/blob/master/src/main/resources/cassandra/creditcard.cql>

Copy the contents of this file and paste it on cqlsh Cassandra client

Create creditcard keyspace

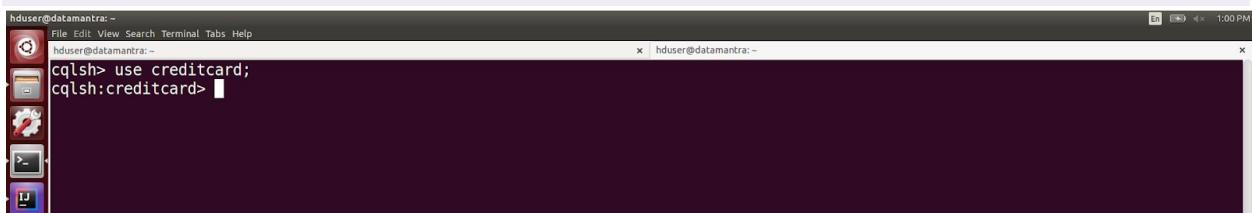
```
CREATE KEYSPACE IF NOT EXISTS creditcard  
WITH replication = {  
  'class': 'SimpleStrategy', 'replication_factor': 1  
};
```



A screenshot of a terminal window titled 'hduser@datamantra:~'. The window shows the command 'CREATE KEYSPACE IF NOT EXISTS creditcard' being typed into the terminal. The command includes 'WITH replication = { ... }' with 'class' set to 'SimpleStrategy' and 'replication_factor' set to 1. The terminal window has a dark background and a light-colored text area. The title bar shows the user 'hduser' and the host 'datamantra'. The status bar at the bottom right indicates the time as 12:59 PM.

Use creditcard keyspace

```
USE creditcard;
```



A screenshot of a terminal window titled 'hduser@datamantra:~'. The command 'use creditcard;' is typed into the terminal. The response 'cqlsh:creditcard>' is displayed, indicating the keyspace has been selected. The terminal window has a dark background and a light-colored text area. The title bar shows the user 'hduser' and the host 'datamantra'. The status bar at the bottom right indicates the time as 1:00 PM.

Create fraud_transaction table

```
CREATE TABLE IF NOT EXISTS fraud_transaction (  
  cc_num text,  
  trans_time timestamp,  
  trans_num text,  
  category text,  
  merchant text,  
  amt double,  
  merch_lat double,  
  merch_long double,  
  distance double,  
  age int,  
  is_fraud double,
```

```
PRIMARY KEY(cc_num, trans_time)
)WITH CLUSTERING ORDER BY (trans_time DESC);
```

A screenshot of a terminal window titled "hduser@datamantra:~". The window shows the command:

```
cqlsh:creditcard> CREATE TABLE IF NOT EXISTS fraud_transaction (
...     cc_num text,
...     trans_time timestamp,
...     trans_num text,
...     category text,
...     merchant text,
...     amt double,
...     merch_lat double,
...     merch_long double,
...     distance double,
...     age int,
...     is_fraud double,
...     PRIMARY KEY(cc_num, trans_time)
... )WITH CLUSTERING ORDER BY (trans_time DESC);
```

The command is being typed into the terminal.

Create not_fraud_transaction table

```
CREATE TABLE IF NOT EXISTS non_fraud_transaction (
    cc_num text,
    trans_time timestamp,
    trans_num text,
    category text,
    merchant text,
    amt double,
    merch_lat double,
    merch_long double,
    distance double,
    age int,
    is_fraud double,
    PRIMARY KEY(cc_num, trans_time)
)WITH CLUSTERING ORDER BY (trans_time DESC);
```

A screenshot of a terminal window titled "hduser@datamantra:~". The window shows the command:

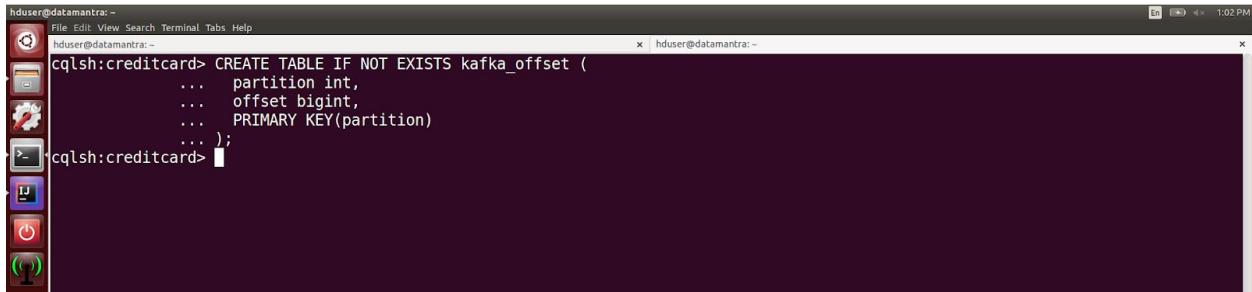
```
cqlsh:creditcard> CREATE TABLE IF NOT EXISTS non_fraud_transaction (
...     cc_num text,
...     trans_time timestamp,
...     trans_num text,
...     category text,
...     merchant text,
...     amt double,
...     merch_lat double,
...     merch_long double,
...     distance double,
...     age int,
...     is_fraud double,
...     PRIMARY KEY(cc_num, trans_time)
... )WITH CLUSTERING ORDER BY (trans_time DESC);
```

The command is being typed into the terminal.

Create a kafka_offset table

Offset per partition will be saved in this table

```
CREATE TABLE IF NOT EXISTS kafka_offset (
    partition int,
    offset bigint,
    PRIMARY KEY(partition)
);
```



A screenshot of a terminal window titled "Terminal". The window shows the command "CREATE TABLE IF NOT EXISTS kafka_offset (" followed by several lines of code. The background of the terminal is dark, and the text is white. The window has a standard OS X style with a title bar and a scroll bar.

Create a customer table

This table will have customer details

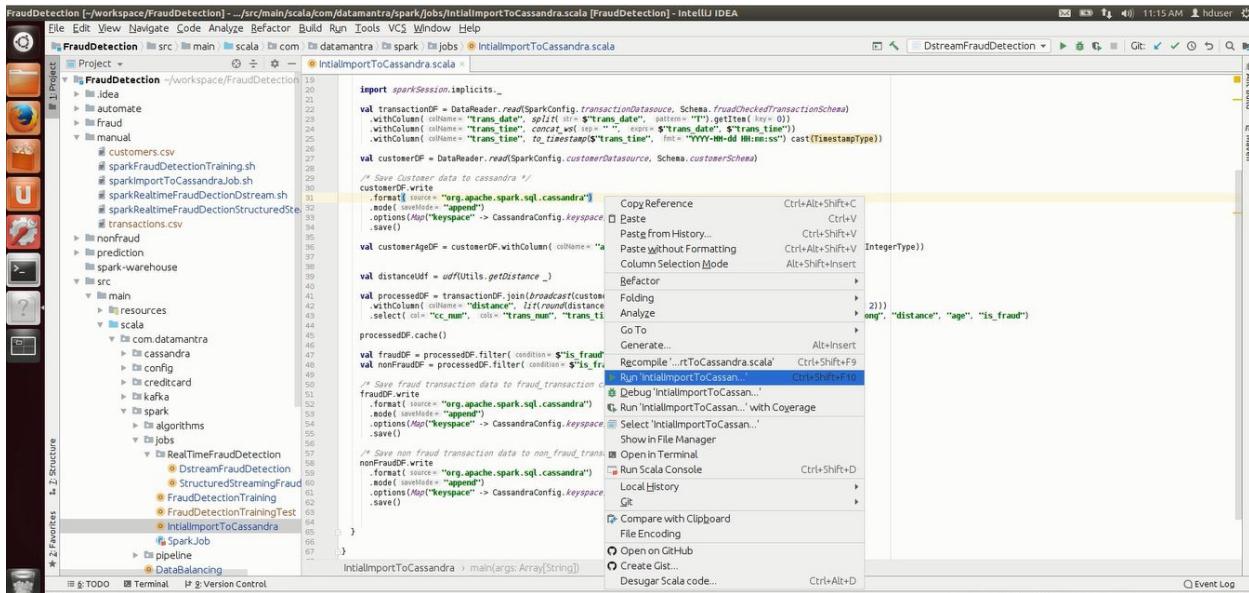
```
CREATE TABLE IF NOT EXISTS customer (
    cc_num text,
    first text,
    last text,
    gender text,
    street text,
    city text,
    state text,
    zip text,
    lat double,
    long double,
    job text,
    dob timestamp,
    PRIMARY KEY(cc_num));
```

```

hduser@datamantra:~ 
File Edit View Search Terminal Tabs Help 
hduser@datamantra:~ 
cqlsh:creditcard> CREATE TABLE IF NOT EXISTS customer (
...     cc_num text,
...     first text,
...     last text,
...     gender text,
...     street text,
...     city text,
...     state text,
...     zip text,
...     lat double,
...     long double,
...     job text,
...     dob timestamp,
...     PRIMARY KEY(cc_num)
... );
cqlsh:creditcard>

```

Run IntialImportToCassandra.scala from IntelliJ No Input parameter



Output

Transaction data and customer data must be saved in the corresponding table in Cassandra

```
SELECT * FROM customer limit 5
```

```
hduser@datamantra:~
```

```
cqlsh:creditcard> SELECT * FROM customer limit 5;

+-----+-----+-----+-----+-----+-----+-----+-----+
| cc_num | city | dob | first | gender | job | last | lat |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 180036251237802 | Salt Lick | 1956-07-19 18:30:00.000000+0000 | Melissa | F | Psychologist, forensic | James | 38.0 |
| 104 | -83.6316 | KY | 537 Bryant Mall | 40371 |
| 676165681542 | Salem | 1954-07-03 18:30:00.000000+0000 | John | M | Human resources officer | Garcia | 44.9 |
| 039 | -123.0445 | OR | 34153 Maria Mountain | 97302 |
| 30157941709315 | Caledonia | 1974-11-03 18:30:00.000000+0000 | Maurice | M | Hydrographic surveyor | Simon | 43.6 |
| 221 | -91.4837 | MN | 031 Jessica Harbor Suite 104 | 55921 |
| 180094108369013 | Zephyr Cove | 1949-12-28 18:30:00.000000+0000 | John | M | Geophysical data processor | Holland | 39.0 |
| 204 | -119.9114 | NV | 630 Christina Harbor | 89448 |
| 4048725581466255 | Chicago | 1950-12-27 18:30:00.000000+0000 | Theresa | F | Retail banker | Cole | 41.8 |
| 858 | -87.6181 | IL | 431 Walker Via | 60601 |

(5 rows)
cqlsh:creditcard>
```

```
SELECT * FROM fraud_transaction limit 5;
```

```
hduser@datamantra:~
```

```
cqlsh:creditcard> SELECT * FROM fraud_transaction limit 5;

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| cc_num | trans_time | age | amt | category | distance | is_fraud | merch_lat | merch_long | merchant | trans_num |
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 180036251237802 | 2012-01-01 17:54:16.000000+0000 | 63 | 2168 | shopping_net | 157.66 | 1 | 37.13679 | -84.94059 | Fisher Inc | f5961217ad20bcb888558e451c0555b8 |
| 180036251237802 | 2012-01-01 17:41:52.000000+0000 | 63 | 2583 | shopping_net | 59.92 | 1 | 38.4867 | -84.11508 | Stamm-Witting | 122b29e9fd69ed43223ad7a56cf38881 |
| 180036251237802 | 2012-01-01 17:28:32.000000+0000 | 63 | 262 | gas_transport | 80.43 | 1 | 38.76958 | -83.27012 | Streich, Hansen and Veun | f5ebbe07eb6fbcf3f194cd24f4b9bf8 |
| 180036251237802 | 2012-01-01 17:07:28.000000+0000 | 63 | 2361 | grocery_pos | 118.8 | 1 | 39.13225 | -84.00291 | M osciski, Gislason and Mertz | f95517fbc6167e00f6adfb3c6c1f1d03 |
| 180036251237802 | 2012-01-01 16:38:01.000000+0000 | 63 | 1723 | grocery_pos | 232.03 | 1 | 36.42305 | -82.07787 | Barton Inc | e8b482a3eb37655c0ce434ca49798458 |

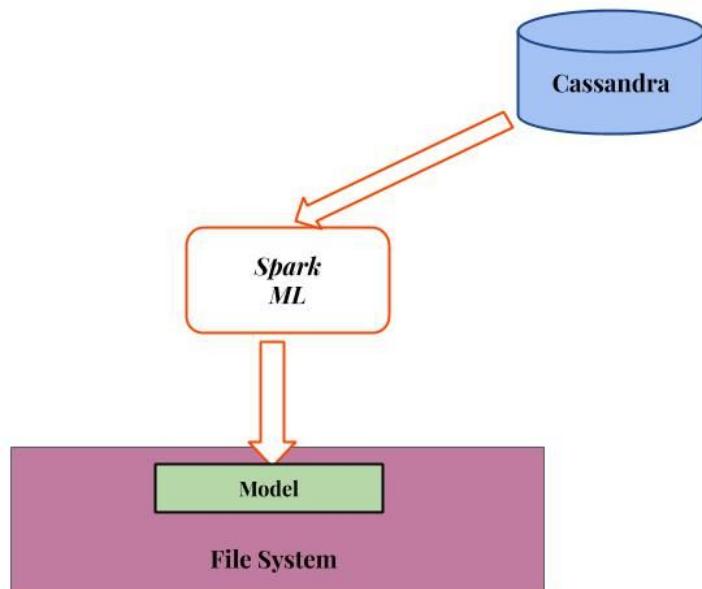
(5 rows)
cqlsh:creditcard>
```

```
SELECT * FROM fraud_transaction limit 5;
```

All the data is successfully imported to Cassandra Keyspace

5. Spark Training Job

5.1 Introduction



This Job will read fraud and non-fraud transaction data from Cassandra.

It will train on this data and it will create a model

This model will be saved to the file system.

Note: There are a few differences between this document and the video. Please consider this document over the video

As said earlier in an actual scenario, Spark ML Job must not be automated. It must be manually run, test and evaluated by the DataScience Team

5.2 Spark Training Steps

5.2.1 Read data from Cassandra

```
val fraudTransactionDF =  
    DataReader.readFromCassandra(CassandraConfig.keyspace,  
        CassandraConfig.fraudTransactionTable)  
        .select("cc_num", "category", "merchant", "distance", "amt", "age",  
        "is_fraud")  
  
val nonFraudTransactionDF =  
    DataReader.readFromCassandra(CassandraConfig.keyspace,  
        CassandraConfig.nonFraudTransactionTable)  
        .select("cc_num", "category", "merchant", "distance", "amt", "age",  
        "is_fraud")  
  
val transactionDF = nonFraudTransactionDF.union(fraudTransactionDF)  
transactionDF.cache()
```

cc_num	category	merchant	distance	amt	age	is_fraud
5157595343543285	gas_transport	Schmitt Inc	0.73	111.0	46	0.0
5157595343543285	kids_pets	Beer-Jast	1.73	66.0	46	0.0
5157595343543285	grocery_pos	Hudson-Ratke	1.23	220.0	46	0.0
5157595343543285	entertainment	Kassulke Inc	2.17	100.0	46	0.0
5157595343543285	shopping_net	Boyer PLC	1.33	80.0	46	0.0
5157595343543285	entertainment	Spencer PLC	1.29	141.0	46	0.0
5157595343543285	entertainment	Brown-Greenholt	0.8	221.0	46	0.0
5157595343543285	travel	Kovacek Ltd	1.5	91.0	46	0.0
5157595343543285	gas_transport	Kutch LLC	0.71	57.0	46	0.0
5157595343543285	kids_pets	Lowe, Dietrich and Erdman	1.41	158.0	46	0.0
5157595343543285	kids_pets	Bednar PLC	1.99	211.0	46	0.0
5157595343543285	kids_pets	Hammes-Beatty	0.88	42.0	46	0.0
5157595343543285	kids_pets	Hilpert-Conroy	0.97	103.0	46	0.0
5157595343543285	gas_transport	Robel, Cummerata and Prosacco	1.26	100.0	46	0.0
5157595343543285	grocery_pos	Strosin-Cruickshank	1.02	75.0	46	0.0
5157595343543285	grocery_pos	Schultz, Simonis and Little	1.8	65.0	46	0.0
5157595343543285	home	Beier and Sons	1.66	112.0	46	0.0
5157595343543285	misc_pos	Turner, Ruecker and Parisian	1.81	27.0	46	0.0
5157595343543285	entertainment	Howe PLC	1.56	151.0	46	0.0
5157595343543285	entertainment	Bauch-Blanda	1.88	126.0	46	0.0

"category", "merchant", "distance", "amt", "age" will be used as features to train the model

The data type of category and merchant columns is String. The data type of distance, amt, and age is Double/Int(Numeric)

5.2.2 String Values to Numeric Values

```
/*String Indexer for the category and merchant columns*/
  val categoryIndexer = new
StringIndexer().setInputCol("category").setOutputCol("category_indexed")
  val merchantIndexer = new
StringIndexer().setInputCol("merchant").setOutputCol("merchant_indexed")
```

StringIndexer transforms string values to double values. Because, ML Algorithms does not understand String values, it understands only numeric values. Input to StringIndexer is "category" and "merchant" column. Output is "category_indexed" and "merchant_indexed" column. StringIndexer will append these 2 columns to the result dataframe. StringIndexer will convert the category and merchant names to double values. As you can see here, 2 new columns are created, "category_indexed" and "merchant_indexed"

cc_num	category	merchant	distance	amt	age	is_fraud	category_indexed	merchant_indexed
5157595343543285	gas_transport	Schmitt Inc	0.73	111.0	46	0.0	0.0	28.0
5157595343543285	kids_pets	Beer-Jast	1.73	66.0	46	0.0	4.0	291.0
5157595343543285	grocery_pos	Hudson-Ratke	1.23	220.0	46	0.0	1.0	310.0
5157595343543285	entertainment	Kassulke Inc	2.17	100.0	46	0.0	9.0	470.0
5157595343543285	shopping_net	Boyer PLC	1.33	80.0	46	0.0	5.0	78.0
5157595343543285	entertainment	Spencer PLC	1.29	141.0	46	0.0	9.0	304.0
5157595343543285	entertainment	Brown-Greenholt	0.8	221.0	46	0.0	9.0	527.0
5157595343543285	travel	Kovacek Ltd	1.5	91.0	46	0.0	13.0	663.0
5157595343543285	gas_transport	Kutch LLC	0.71	57.0	46	0.0	0.0	2.0
5157595343543285	kids_pets	Lowe, Dietrich and Erdman	1.41	158.0	46	0.0	4.0	230.0
5157595343543285	kids_pets	Bednar PLC	1.99	211.0	46	0.0	4.0	101.0
5157595343543285	kids_pets	Hammes-Beatty	0.88	42.0	46	0.0	4.0	242.0
5157595343543285	kids_pets	Hilpert-Conroy	0.97	103.0	46	0.0	4.0	223.0
5157595343543285	gas_transport	Robel, Cummerata and Prosacco	1.26	100.0	46	0.0	0.0	32.0
5157595343543285	grocery_pos	Strosin-Cruickshank	1.02	75.0	46	0.0	1.0	181.0
5157595343543285	grocery_pos	Schultz, Simonis and Little	1.8	65.0	46	0.0	1.0	133.0
5157595343543285	home	Beier and Sons	1.66	112.0	46	0.0	3.0	81.0
5157595343543285	misc_pos	Turner, Ruecker and Parisian	1.81	27.0	46	0.0	7.0	229.0
5157595343543285	entertainment	Howe PLC	1.56	151.0	46	0.0	9.0	201.0
5157595343543285	entertainment	Bauch-Blanda	1.88	126.0	46	0.0	9.0	509.0

only showing top 20 rows

5.2.3 Scaling Numeric Columns

The main idea is to Normalize/Standardize (`mean = 0` and `standard deviation = 1`) your features before applying machine learning techniques.

`StandardScaler` does not work on double values. Hence, first we need to convert all 3 columns into vector, then slice it and finally scale it.

```
/*Transform raw numerical columns to vector. Slice the vector and Scale the
vector. Scaling is required so that all the column values are at the same
level of measurement */
val numericAssembler = new VectorAssembler().setInputCols(Array("distance",
"amt", "age" )).setOutputCol("rawfeature")
val slicer = new
VectorSlicer().setInputCol("rawfeature").setOutputCol("slicedfeatures").set
Names(Array("distance", "amt", "age"))
val scaler = new
StandardScaler().setInputCol("slicedfeatures").setOutputCol("scaledfeatures"
")
```

VectorAssembler

```
val numericAssembler = new VectorAssembler().setInputCols(Array("distance",
"amt", "age" )).setOutputCol("rawfeature")
```

`VectorAssembler` is used to transform multiple column values into a single vector column. Input to `VectorAssembler` is “distance”, “amt” and “age” columns. The output is the “rawfeature” column. It is a vector column.

distance	amt	age	is_fraud	category_indexed	merchant_indexed	rawfeature
0.73	111.0	46	0.0	0.0	28.0	[0.73, 111.0, 46.0]
1.73	66.0	46	0.0	4.0	291.0	[1.73, 66.0, 46.0]
1.23	220.0	46	0.0	1.0	310.0	[1.23, 220.0, 46.0]
2.17	100.0	46	0.0	9.0	470.0	[2.17, 100.0, 46.0]
1.33	80.0	46	0.0	5.0	78.0	[1.33, 80.0, 46.0]
1.29	141.0	46	0.0	9.0	304.0	[1.29, 141.0, 46.0]
0.8	221.0	46	0.0	9.0	527.0	[0.8, 221.0, 46.0]
1.5	91.0	46	0.0	13.0	663.0	[1.5, 91.0, 46.0]
0.71	57.0	46	0.0	0.0	2.0	[0.71, 57.0, 46.0]
1.41	158.0	46	0.0	4.0	230.0	[1.41, 158.0, 46.0]
1.99	211.0	46	0.0	4.0	101.0	[1.99, 211.0, 46.0]
0.88	42.0	46	0.0	4.0	242.0	[0.88, 42.0, 46.0]
0.97	103.0	46	0.0	4.0	223.0	[0.97, 103.0, 46.0]
1.26	100.0	46	0.0	0.0	32.0	[1.26, 100.0, 46.0]
1.02	75.0	46	0.0	1.0	181.0	[1.02, 75.0, 46.0]
1.8	65.0	46	0.0	1.0	133.0	[1.8, 65.0, 46.0]
1.66	112.0	46	0.0	3.0	81.0	[1.66, 112.0, 46.0]
1.81	27.0	46	0.0	7.0	229.0	[1.81, 27.0, 46.0]
1.56	151.0	46	0.0	9.0	201.0	[1.56, 151.0, 46.0]
1.88	126.0	46	0.0	9.0	509.0	[1.88, 126.0, 46.0]

VectorSlicer

```
val slicer = new
VectorSlicer().setInputCol("rawfeature").setOutputCol("slicedfeatures").setNa
mes(Array("distance", "amt", "age"))
```

VectorSlicer will slice a single vector into multiple vectors. Input to VectorSlicer is the output of VectorAssembler i.e “rawfeature” column. The output of VectorSlicer is the “slicedfeatures” column. This column contains “distance”, “amt” and “age” as individual vectors.

distance	amt	age	is_fraud	category_indexed	merchant_indexed	rawfeature	slicedfeatures	
0.73	111.0	46	0.0	0.0	28.0	[[0.73, 111.0, 46.0]]	[0.73, 111.0, 46.0]	
1.73	66.0	46	0.0	4.0	291.0	[[1.73, 66.0, 46.0]]	[1.73, 66.0, 46.0]	
1.23	220.0	46	0.0	1.0	310.0	[[1.23, 220.0, 46.0]]	[1.23, 220.0, 46.0]	
2.17	100.0	46	0.0	9.0	470.0	[[2.17, 100.0, 46.0]]	[2.17, 100.0, 46.0]	
1.33	80.0	46	0.0	5.0	78.0	[[1.33, 80.0, 46.0]]	[1.33, 80.0, 46.0]	
1.29	141.0	46	0.0	9.0	304.0	[[1.29, 141.0, 46.0]]	[1.29, 141.0, 46.0]	
0.8	221.0	46	0.0	9.0	527.0	[[0.8, 221.0, 46.0]]	[0.8, 221.0, 46.0]	
1.5	91.0	46	0.0	13.0	663.0	[[1.5, 91.0, 46.0]]	[1.5, 91.0, 46.0]	
0.71	57.0	46	0.0	0.0	2.0	[[0.71, 57.0, 46.0]]	[0.71, 57.0, 46.0]	
1.41	158.0	46	0.0	4.0	230.0	[[1.41, 158.0, 46.0]]	[1.41, 158.0, 46.0]	
1.99	211.0	46	0.0	4.0	101.0	[[1.99, 211.0, 46.0]]	[1.99, 211.0, 46.0]	
0.88	42.0	46	0.0	4.0	242.0	[[0.88, 42.0, 46.0]]	[0.88, 42.0, 46.0]	
0.97	103.0	46	0.0	4.0	223.0	[[0.97, 103.0, 46.0]]	[0.97, 103.0, 46.0]	
co	1.26	100.0	46	0.0	0.0	32.0	[[1.26, 100.0, 46.0]]	[1.26, 100.0, 46.0]
1.02	75.0	46	0.0	1.0	181.0	[[1.02, 75.0, 46.0]]	[1.02, 75.0, 46.0]	
1.8	65.0	46	0.0	1.0	133.0	[[1.8, 65.0, 46.0]]	[1.8, 65.0, 46.0]	
1.66	112.0	46	0.0	3.0	81.0	[[1.66, 112.0, 46.0]]	[1.66, 112.0, 46.0]	
n	1.81	27.0	46	0.0	7.0	229.0	[[1.81, 27.0, 46.0]]	[1.81, 27.0, 46.0]
1.56	151.0	46	0.0	9.0	201.0	[[1.56, 151.0, 46.0]]	[1.56, 151.0, 46.0]	
1.88	126.0	46	0.0	9.0	509.0	[[1.88, 126.0, 46.0]]	[1.88, 126.0, 46.0]	

StandardScaler

```
val scaler = new
StandardScaler().setInputCol("slicedfeatures").setOutputCol("scaledfeatures")
```

The main idea is to Normalize/Standardize (mean = 0 and standard deviation = 1) your features before applying machine learning techniques.

Input to StandardScaler is the output of the VectorSlicer i.e “slicedfeatures” column. The output of StandardScaler is the “scaledfeatures” column. Now this column contains all normalized values

distance	amt	age	is_fraud	category_indexed	merchant_indexed	rawfeature	slicedfeatures	scaledfeatures
0.73	111.0	46	0.0	0.0	28.0	[[0.73, 111.0, 46.0]]	[0.73, 111.0, 46.0]	[0.269256273297367, 0.5987775554848228, 4, 5503973019512705]
1.73	66.0	46	0.0	4.0	291.0	[[1.73, 66.0, 46.0]]	[1.73, 66.0, 46.0]	[0.06381004832937602, 0.35602989785584055, 4, 5503973019512705]
1.23	220.0	46	0.0	1.0	310.0	[[1.23, 220.0, 46.0]]	[1.23, 220.0, 46.0]	[0.1867663261861352, 4, 5503973019512705]
2.17	100.0	46	0.0	9.0	470.0	[[2.17, 100.0, 46.0]]	[2.17, 100.0, 46.0]	[0.08003919356921732, 0.539439239175516, 4, 5503973019512705]
1.33	80.0	46	0.0	5.0	78.0	[[1.33, 80.0, 46.0]]	[1.33, 80.0, 46.0]	[0.049056279929520295, 0.431551391404128, 4, 5503973019512705]
1.29	141.0	46	0.0	9.0	304.0	[[1.29, 141.0, 46.0]]	[1.29, 141.0, 46.0]	[0.04758090308953472, 0.7606093272374775, 4, 5503973019512705]
0.8	221.0	46	0.0	9.0	527.0	[[0.8, 221.0, 46.0]]	[0.8, 221.0, 46.0]	[0.029507536799711454, 1, 1921607185778904, 4, 5503973019512705]
1.5	91.0	46	0.0	13.0	663.0	[[1.5, 91.0, 46.0]]	[1.5, 91.0, 46.0]	[0.055326631499458975, 0.49088970764971956, 4, 5503973019512705]
0.71	57.0	46	0.0	0.0	2.0	[[0.71, 57.0, 46.0]]	[0.71, 57.0, 46.0]	[0.026187938909743913, 0, 3074803663300441, 4, 5503973019512705]
1.41	158.0	46	0.0	4.0	230.0	[[1.41, 158.0, 46.0]]	[1.41, 158.0, 46.0]	[0.05200703360949143, 0.8523139978973153, 4, 5503973019512705]
1.99	211.0	46	0.0	4.0	101.0	[[1.99, 211.0, 46.0]]	[1.99, 211.0, 46.0]	[0.0733999778922825, 1, 1382167946603388, 4, 5503973019512705]
0.88	42.0	46	0.0	4.0	242.0	[[0.88, 42.0, 46.0]]	[0.88, 42.0, 46.0]	[0.0324582904796826, 0.22656448045371672, 4, 5503973019512705]
0.97	103.0	46	0.0	4.0	223.0	[[0.97, 103.0, 46.0]]	[0.97, 103.0, 46.0]	[0.03577788836965014, 0.556224163507815, 4, 5503973019512705]
1.26	100.0	46	0.0	0.0	32.0	[[1.26, 100.0, 46.0]]	[1.26, 100.0, 46.0]	[0.04647437045954554, 0.539439239175516, 4, 5503973019512705]
1.02	75.0	46	0.0	1.0	181.0	[[1.02, 75.0, 46.0]]	[1.02, 75.0, 46.0]	[0.0376221094196321, 0, 404579429381637, 4, 5503973019512705]
1.8	65.0	46	0.0	1.0	133.0	[[1.8, 65.0, 46.0]]	[1.8, 65.0, 46.0]	[0.06639195779935077, 0, 350635505460854, 4, 5503973019512705]
1.66	112.0	46	0.0	3.0	81.0	[[1.66, 112.0, 46.0]]	[1.66, 112.0, 46.0]	[0.0612281385940126, 0.6041719478765779, 4, 5503973019512705]
1.81	27.0	46	0.0	7.0	229.0	[[1.81, 27.0, 46.0]]	[1.81, 27.0, 46.0]	[0.06676080209934717, 0, 14564885457738932, 4, 5503973019512705]
1.56	151.0	46	0.0	9.0	201.0	[[1.56, 151.0, 46.0]]	[1.56, 151.0, 46.0]	[0.057539696759437334, 0.8145532511550292, 4, 5503973019512705]
1.88	126.0	46	0.0	9.0	509.0	[[1.88, 126.0, 46.0]]	[1.88, 126.0, 46.0]	[0.0693427114932192, 0.679693441361502, 4, 5503973019512705]

5.2.4 Feature Column

Now all the feature columns has to be transformed into single feature column. i.e “category_index”, “merchant_index” and “scaledfeatures” columns into single “feature” column.

```
val vectorAssembler = new  
VectorAssembler().setInputCols(Array("category_indexed",  
"merchant_indexed", "scaledfeatures")).setOutputCol("features")
```

One more VectorAssembler is created. Input to this VectorAssembler is the output of previous transformations i.e. “category_indexed”, “merchant_indexed” and “scaledfeatures” columns. Output is “feature column”. All the columns are combined as a single column of vector data type.

category_indexed	merchant_indexed	rawfeature	slicedfeatures	scaledfeatures	features
0.0	[28.0	[0, 73, 111, 0, 46, 0]	[0, 73, 111, 0, 46, 0]	[0, 0269256273297367, 0, 589777554846228, 4, 5503973019512705]	[0, 0, 28, 0, 0, 0269256273297367, 0, 589777554846228, 4, 5503973019512705]
14.0	[291.0	[1, 73, 66, 0, 46, 0]	[1, 73, 66, 0, 46, 0]	[0, 06530104832937602, 0, 25602989705894055, 4, 5503973019512705]	[1, 0, 291, 0, 0, 06530104832937602, 0, 25602989705894055, 4, 5503973019512705]
11.0	[310.0	[1, 23, 220, 0, 46, 0]	[1, 23, 220, 0, 46, 0]	[0, 04536738782955636, 1, 1867663261961352, 4, 5503973019512705]	[1, 0, 310, 0, 0, 04536738782955636, 1, 1867663261961352, 4, 5503973019512705]
19.0	[470.0	[2, 17, 100, 0, 46, 0]	[2, 17, 100, 0, 46, 0]	[0, 06003919356921732, 0, 539439239175516, 4, 5503973019512705]	[1, 0, 470, 0, 0, 0, 06003919356921732, 0, 539439239175516, 4, 5503973019512705]
15.0	[78.0	[1, 33, 80, 0, 46, 0]	[1, 33, 80, 0, 46, 0]	[0, 049056729929520295, 0, 4315513913404128, 4, 5503973019512705]	[1, 0, 78, 0, 0, 049056729929520295, 0, 4315513913404128, 4, 5503973019512705]
19.0	[304.0	[1, 29, 141, 0, 46, 0]	[1, 29, 141, 0, 46, 0]	[0, 04758090308953472, 0, 760609327274775, 4, 5503973019512705]	[1, 0, 304, 0, 0, 04758090308953472, 0, 760609327274775, 4, 5503973019512705]
19.0	[527.0	[0, 8, 221, 0, 46, 0]	[0, 8, 221, 0, 46, 0]	[0, 029507536799711454, 1, 1921607185778904, 4, 5503973019512705]	[1, 0, 527, 0, 0, 0, 029507536799711454, 1, 1921607185778904, 4, 5503973019512705]
13.0	[663.0	[1, 5, 91, 0, 46, 0]	[1, 5, 91, 0, 46, 0]	[0, 05532631499458975, 0, 4908897076497195, 4, 5503973019512705]	[1, 0, 663, 0, 0, 05532631499458975, 0, 4908897076497195, 4, 5503973019512705]
0.0	[2.0	[0, 71, 57, 0, 46, 0]	[0, 71, 57, 0, 46, 0]	[0, 02618793699743913, 0, 3074803663300441, 4, 5503973019512705]	[1, 0, 2, 0, 0, 02618793699743913, 0, 3074803663300441, 4, 5503973019512705]
4.0	[230.0	[1, 41, 158, 0, 46, 0]	[1, 41, 158, 0, 46, 0]	[0, 05200703360919143, 0, 8523139978973153, 4, 5503973019512705]	[1, 0, 230, 0, 0, 05200703360919143, 0, 8523139978973153, 4, 5503973019512705]
4.0	[100.0	[1, 99, 218, 0, 46, 0]	[1, 99, 218, 0, 46, 0]	[0, 0733999977892225, 0, 8523139978973153, 4, 5503973019512705]	[1, 0, 101, 0, 0, 0733999977892225, 0, 8523139978973153, 4, 5503973019512705]
4.0	[242.0	[1, 08, 42, 0, 46, 0]	[1, 08, 42, 0, 46, 0]	[0, 0324582604796828, 0, 22861640845371672, 4, 5503973019512705]	[1, 0, 242, 0, 0, 0324582604796828, 0, 22861640845371672, 4, 5503973019512705]
4.0	[223.0	[0, 97, 100, 0, 46, 0]	[0, 97, 100, 0, 46, 0]	[0, 0556224163502154, 0, 5556224163502154, 4, 5503973019512705]	[1, 0, 223, 0, 0, 0556224163502154, 0, 5556224163502154, 4, 5503973019512705]
0.0	[32.0	[1, 26, 100, 0, 46, 0]	[1, 26, 100, 0, 46, 0]	[0, 04647437045954554, 0, 53943923017516, 4, 5503973019512705]	[1, 0, 32, 0, 0, 04647437045954554, 0, 53943923017516, 4, 5503973019512705]
1.0	[181.0	[1, 02, 75, 0, 46, 0]	[1, 02, 75, 0, 46, 0]	[0, 0376221094196321, 0, 404573429381637, 4, 5503973019512705]	[1, 0, 181, 0, 0, 0376221094196321, 0, 404573429381637, 4, 5503973019512705]
1.0	[133.0	[1, 8, 65, 0, 46, 0]	[1, 8, 65, 0, 46, 0]	[0, 06639195779935077, 0, 3506355054640954, 4, 5503973019512705]	[1, 0, 133, 0, 0, 06639195779935077, 0, 3506355054640954, 4, 5503973019512705]
3.0	[81.0	[1, 66, 112, 0, 46, 0]	[1, 66, 112, 0, 46, 0]	[0, 06122813885940126, 0, 6041719478765779, 4, 5503973019512705]	[1, 0, 81, 0, 0, 06122813885940126, 0, 6041719478765779, 4, 5503973019512705]
7.0	[229.0	[1, 81, 27, 0, 46, 0]	[1, 81, 27, 0, 46, 0]	[0, 06676080200934717, 0, 14564859457738932, 4, 5503973019512705]	[1, 0, 229, 0, 0, 06676080200934717, 0, 14564859457738932, 4, 5503973019512705]
9.0	[201.0	[1, 56, 151, 0, 46, 0]	[1, 56, 151, 0, 46, 0]	[0, 057539696759437334, 0, 8145532511550292, 4, 5503973019512705]	[1, 0, 201, 0, 0, 057539696759437334, 0, 8145532511550292, 4, 5503973019512705]
9.0	[509.0	[1, 88, 126, 0, 46, 0]	[1, 88, 126, 0, 46, 0]	[0, 06934271147932192, 0, 679693413611502, 4, 5503973019512705]	[1, 0, 509, 0, 0, 06934271147932192, 0, 679693413611502, 4, 5503973019512705]

5.2.5 Pipeline Stages

A Spark Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. These stages are run in order, and the input DataFrame is transformed as it passes through each stage.

A Transformer is an algorithm that transforms one DataFrame to another by using the `transform()` method. For example, a feature transformer could read one column of a DataFrame, map it to another column, and output a new DataFrame with the mapped column appended to it.

An Estimator is an abstraction of learning algorithms and is responsible for fitting or training on a dataset to produce a Transformer. An Estimator implements a method named `fit()`, which accepts a DataFrame and produces a Model, which is a

Transformer.

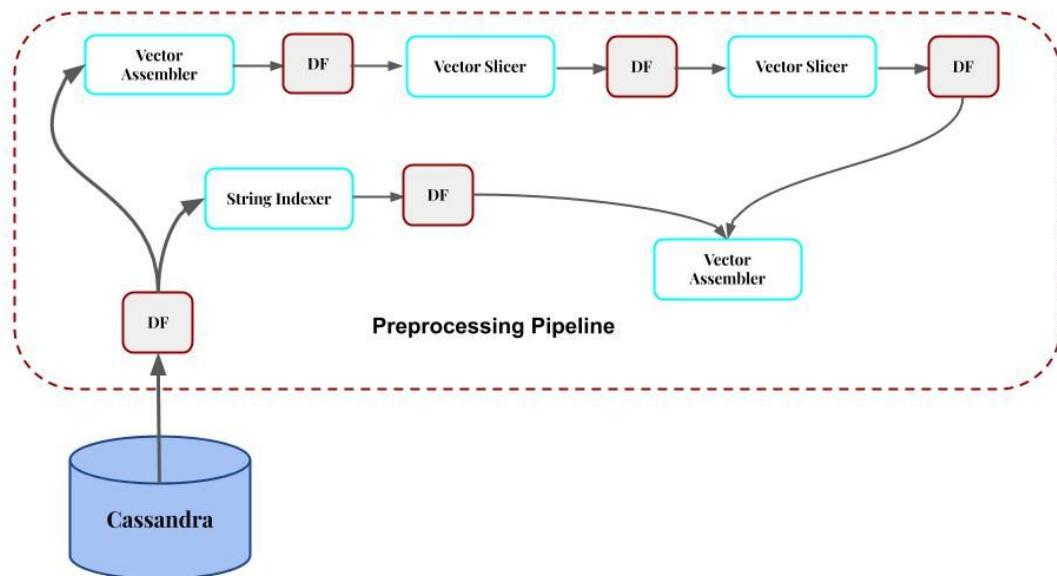
StringIndexer is an Estimator.

VectorAssembler is a Transformer

VectorSlicer is a Transformer

StandarScaler is an Estimator

Preprocessing Pipeline Model



```
val pipeline = new Pipeline().setStages(Array(categoryIndexer,  
merchantIndexer, numericAssembler, slicer, scaler, vectorAssembler))  
val preprocessingTransformerModel = pipeline.fit(transactionDF)  
val featureDF = preprocessingTransformerModel.transform(transactionDF)
```

Here we are building preprocessing pipeline with our categoryIndexer, merchantIndexer, numericAssembler, slicer, scaler, and vectorAssembler.

fit() method returns a Model which is a Transformer

On calling transform() method on the preprocessingTransformerModel, transform method of all the stages are called in the order specified in the pipeline. As a

result, new columns will be added. All the output snapshots that we specified earlier for each individual Stage will actually happen on calling `transform()` method.

category_indexed	merchant_indexed	rawfeature	slicedfeatures	scaledfeatures	features
0.0	28.0	[10, 73, 112, 0, 46, 0]	[10, 73, 112, 0, 46, 0]	[0, 0.26925373293767, 0, 567777554846228, 4, 5503973019512705]	[10, 0, 2, 0, 0, 0.026925373293767, 0, 567777554846228, 4, 5503973019512705]
4.0	291.0	[11, 73, 66, 0, 46, 0]	[11, 73, 66, 0, 46, 0]	[0, 0.0680404322677602, 0, 567777554846228, 4, 5503973019512705]	[11, 0, 291, 0, 0, 0.0680404322677602, 0, 567777554846228, 4, 5503973019512705]
1.0	310.0	[12, 23, 220, 0, 46, 0]	[12, 23, 220, 0, 46, 0]	[0, 0.04536703295621705, 1, 1.067662361801352, 4, 5503973019512705]	[12, 0, 310, 0, 0, 0.04536703295621705, 1, 1.067662361801352, 4, 5503973019512705]
19.0	470.0	[12, 17, 100, 0, 46, 0]	[12, 17, 100, 0, 46, 0]	[0, 0.0800391056921732, 0, 53043923917516, 4, 5503973019512705]	[19, 0, 470, 0, 0, 0.0800391056921732, 0, 53043923917516, 4, 5503973019512705]
5.0	178.0	[11, 33, 80, 0, 46, 0]	[11, 33, 80, 0, 46, 0]	[0, 0.04956279929520295, 0, 421513913404128, 4, 5503973019512705]	[15, 0, 78, 0, 0, 0.04956279929520295, 0, 421513913404128, 4, 5503973019512705]
9.0	304.0	[11, 29, 141, 0, 46, 0]	[11, 29, 141, 0, 46, 0]	[0, 0.04758090308953472, 0, 7606093272374775, 4, 5503973019512705]	[19, 0, 304, 0, 0, 0.04758090308953472, 0, 7606093272374775, 4, 5503973019512705]
9.0	527.0	[10, 8, 221, 0, 46, 0]	[10, 8, 221, 0, 46, 0]	[0, 0.029507536799711454, 1, 1.1921607185778904, 4, 5503973019512705]	[19, 0, 527, 0, 0, 0.029507536799711454, 1, 1.1921607185778904, 4, 5503973019512705]
13.0	663.0	[11, 5, 91, 0, 46, 0]	[11, 5, 91, 0, 46, 0]	[0, 0.055326631499458975, 0, 49088970764971956, 4, 5503973019512705]	[13, 0, 663, 0, 0, 0.055326631499458975, 0, 49088970764971956, 4, 5503973019512705]
0.0	2.0	[0, 71, 57, 0, 46, 0]	[0, 71, 57, 0, 46, 0]	[0, 0.0261879389749313, 0, 3074803663300441, 4, 5503973019512705]	[10, 0, 2, 0, 0, 0.0261879389749313, 0, 3074803663300441, 4, 5503973019512705]
4.0	230.0	[11, 41, 158, 0, 46, 0]	[11, 41, 158, 0, 46, 0]	[0, 0.0520070336909443, 0, 4523139978973153, 4, 5503973019512705]	[14, 0, 230, 0, 0, 0.0520070336909443, 0, 4523139978973153, 4, 5503973019512705]
4.0	101.0	[11, 99, 211, 0, 46, 0]	[11, 99, 211, 0, 46, 0]	[0, 0.0733999778928225, 1, 1.38216794663386, 4, 5503973019512705]	[14, 0, 101, 0, 0, 0.0733999778928225, 1, 1.38216794663386, 4, 5503973019512705]
4.0	242.0	[10, 88, 42, 0, 46, 0]	[10, 88, 42, 0, 46, 0]	[0, 0.032458294798626, 0, 22656448045371672, 4, 5503973019512705]	[14, 0, 242, 0, 0, 0.032458294798626, 0, 22656448045371672, 4, 5503973019512705]
4.0	223.0	[10, 97, 103, 0, 46, 0]	[10, 97, 103, 0, 46, 0]	[0, 0.028777445965844, 1, 1.2432416287815, 4, 5503973019512705]	[14, 0, 223, 0, 0, 0.028777445965844, 1, 1.2432416287815, 4, 5503973019512705]
0.0	32.0	[11, 26, 100, 0, 46, 0]	[11, 26, 100, 0, 46, 0]	[0, 0.042877045624554, 0, 53043923917516, 4, 5503973019512705]	[10, 0, 32, 0, 0, 0.042877045624554, 0, 53043923917516, 4, 5503973019512705]
1.0	181.0	[11, 66, 75, 0, 46, 0]	[11, 66, 75, 0, 46, 0]	[0, 0.0376221094196321, 0, 404579423931637, 4, 5503973019512705]	[11, 0, 181, 0, 0, 0.0376221094196321, 0, 404579423931637, 4, 5503973019512705]
1.0	133.0	[11, 8, 65, 0, 46, 0]	[11, 8, 65, 0, 46, 0]	[0, 0.0663010577995077, 0, 2506355054640854, 4, 5503973019512705]	[11, 0, 133, 0, 0, 0.0663010577995077, 0, 2506355054640854, 4, 5503973019512705]
3.0	81.0	[11, 66, 112, 0, 46, 0]	[11, 66, 112, 0, 46, 0]	[0, 0.06122813885940126, 0, 6041719478765779, 4, 5503973019512705]	[13, 0, 81, 0, 0, 0.06122813885940126, 0, 6041719478765779, 4, 5503973019512705]
7.0	229.0	[11, 81, 27, 0, 46, 0]	[11, 81, 27, 0, 46, 0]	[0, 0.066760080200934717, 0, 1.456485457738932, 4, 5503973019512705]	[17, 0, 229, 0, 0, 0.066760080200934717, 0, 1.456485457738932, 4, 5503973019512705]
9.0	201.0	[11, 56, 151, 0, 46, 0]	[11, 56, 151, 0, 46, 0]	[0, 0.057539696759437334, 0, 814532511550292, 4, 5503973019512705]	[19, 0, 201, 0, 0, 0.057539696759437334, 0, 814532511550292, 4, 5503973019512705]
9.0	509.0	[11, 88, 126, 0, 46, 0]	[11, 88, 126, 0, 46, 0]	[0, 0.06934271147932192, 0, 679693413611502, 4, 5503973019512705]	[19, 0, 509, 0, 0, 0.06934271147932192, 0, 679693413611502, 4, 5503973019512705]

5.2.6 Split data

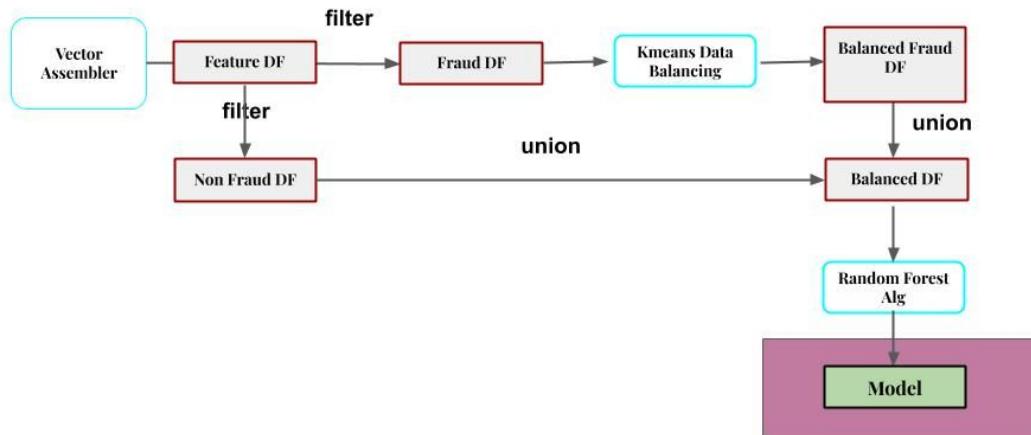
Split data into training and testing. Training data is 80% and Test data is 20%

```
val Array(trainData, testData) = featureDF.randomSplit(Array(0.8, 0.2))
```

5.2.7 Data Balancing

There will be usually more genuine transactions(millions) then fraud transactions(a few hundred). Such data is imbalanced data and must be balanced. So genuine transactions must be balanced. No. of genuine transactions are balanced(reduced) to no. of fraud transaction. Here we are using K Means Algorithm to balance genuine transactions. There are other ways to balance data.

No. of K-means centroid = no. of fraud transaction



```
object DataBalancing {

    val logger = Logger.getLogger(getClass.getName)

    def createBalancedDataframe(df:DataFrame, reductionCount:Int)(implicit sparkSession:SparkSession) = {

        val kMeans = new KMeans().setK(reductionCount).setMaxIter(30)
        val kMeansModel = kMeans.fit(df)

        import sparkSession.implicits._
        kMeansModel.clusterCenters.toList.map(v => (v, 0)).toDF("features",
        "is_fraud")
    }
}
```

```
val featureLabelDF = trainData.select("features", "is_fraud").cache()
val nonFraudDF = featureLabelDF.filter($"is_fraud" === 0)
val fraudDF = featureLabelDF.filter($"is_fraud" === 1)
val fraudCount = fraudDF.count()

val balancedNonFraudDF = DataBalancing.createBalancedDataframe(nonFraudDF,
fraudCount.toInt)
val finalfeatureDF = fraudDF.union(balancedNonFraudDF)
```

Output of the pipeline is a DataFrame which along with other columns contains a “feature” column and “is_fraud” column. Values of “is_fraud” column is either 1(fraud) or 0(non-fraud). We will select only these 2 columns as the training algorithm requires only these 2 columns.

Next, we separate fraud transactions from non-fraud transactions as 2 different dataframe.

Non-fraud transactions DataFrame is fed to the K-Means algorithm to balance the dataframe. K-Means Algorithm reduces no. of non-fraud transactions to no. of fraud transactions.

Output Dataframe of K-means is unioned with the fraud transaction dataframe.

Now we have a balanced dataframe.

5.2.8 Training

This is a classification problem. Either fraud or non-fraud. Here we are using Random Forest. In a real scenario, you must try different Classification algorithms. Evaluate these algorithms using confusion matrix, AUC, ROC, etc. But here we are not evaluating different algorithms. We are using Random Forest.

```
def randomForestClassifier(df: org.apache.spark.sql.DataFrame)(implicit
  sparkSession: SparkSession) = {
  import sparkSession.implicits._
  val randomForestEstimator = new
    RandomForestClassifier().setLabelCol("is_fraud").setFeaturesCol("features").setMax
    Bins(700)
  val model = randomForestEstimator.fit(df)
  model
}
```

```
val randomForestModel = Algorithms.randomForestClassifier(finalfeatureDF)
val predictionDF = randomForestModel.transform(testData)
predictionDF.show(false)
```

RandomForest is an estimator. Hence we have to call fit() method. On calling the fit() method, the Random Forest algorithm will actually start training on the data and will

finally return a model, which is a transformer.

This is our Random Forest Model. Using this model we are predicting the test data by calling the transform() method.

features	rawPrediction	probability	prediction
[5.0, 337.0, 1.7807798458625863, 8.690366143117563, 7.616969396744517]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[1.0, 79.0, 5.554056114125689, 1.3270205283717693, 4.5503973019512705]	[[1.0, 19.0]]	[[0.05, 0.95]]	1.0
[3.0, 239.0, 3.6364350663544402, 2.562336386083701, 4.5503973019512705]	[[1.0, 19.0]]	[[0.05, 0.95]]	1.0
[2.0, 18.0, 6.1253957954101015, 2.492209284990884, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[13.0, 628.0, 7.167011844439916, 1.5104298696914449, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[13.0, 604.0, 6.207648054239297, 2.524575639341415, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[9.0, 519.0, 5.412051093277077, 3.0748036633004414, 4.253632260519666]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[2.0, 86.0, 6.791159594453592, 3.9217232688060015, 4.253632260519666]	[[2.0, 18.0]]	[[0.1, 0.9]]	1.0
[1.0, 65.0, 5.718929475994076, 2.9830989926406035, 6.528830911495301]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[5.0, 382.0, 2.5649426363149184, 5.000601747157034, 6.528830911495301]	[[2.0, 18.0]]	[[0.1, 0.9]]	1.0
[2.0, 109.0, 4.640797850174619, 2.1038130327845126, 6.528830911495301]	[[1.0, 19.0]]	[[0.05, 0.95]]	1.0
[13.0, 655.0, 7.919822877042554, 3.3445232828881992, 6.528830911495301]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[4.0, 223.0, 3.314434071027589, 1.5427962240419757, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[11.0, 476.0, 5.995931477701367, 2.805084043712683, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[1.0, 69.0, 3.771800891423117, 1.6668672490523444, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[11.0, 494.0, 5.026977738040842, 1.693839211011202, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[2.0, 215.0, 6.829888236503212, 1.6560784642688342, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[2.0, 174.0, 5.0, 0.095951605310168, 2.8482391828467244, 4.5503973019512705]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[1.0, 133.0, 0.0, 0.4761778751053436, 3.064014878516931, 7.023439313881308]	[[0.0, 20.0]]	[[0.0, 1.0]]	1.0
[5.0, 177.0, 4.1314239961696, 3.1179588024344826, 7.023439313881308]	[[1.0, 19.0]]	[[0.05, 0.95]]	1.0

5.2.9 Confusion Matrix

```
===== Confusion matrix =====
#####| Actual = 1           Actual = 0
-----+
Predicted = 1| 91.000000      83.000000
Predicted = 0| 0.000000      2548.000000
-----
True Positive Rate: 1.0
False Positive Rate: 0.03154694
Precision: 0.5229885
```

TruePositive(TP) = Model predicting fraud transaction as fraud = 91

False Positive(FP) = Model predicting genuine transactions as fraud = 83

False Negative(FN) = Model predicting fraud transactions as genuine = 0

True Negative(TN) = Model predicting genuine transaction as genuine = 2548

Out of all 91 fraud transactions, the model has correctly predicted all 91 fraud transaction as fraud.

Out of 2631 genuine transactions, the model has wrongly predicted 83 as fraud transactions and correctly predicted 2548 as genuine transactions.

True Positive Rate = $\frac{TP}{TP + FN} = \frac{91.0}{91.0 + 0} = 1.0$

Out of all the fraud transactions, how much we predicted correctly, It should be high as possible.

False Positive Rate (FPR) = $\frac{FP}{FP + TN}$

Out of all the genuine transactions(not fraud), how much we predicted wrong(predicted as fraud). It should be low as possible

Precision = $\frac{TP}{TP + FP} = \frac{91}{91.0 + 83.0} = 0.5229885$

Out of all the fraud transactions that we have predicted correctly, how many are actually fraud.

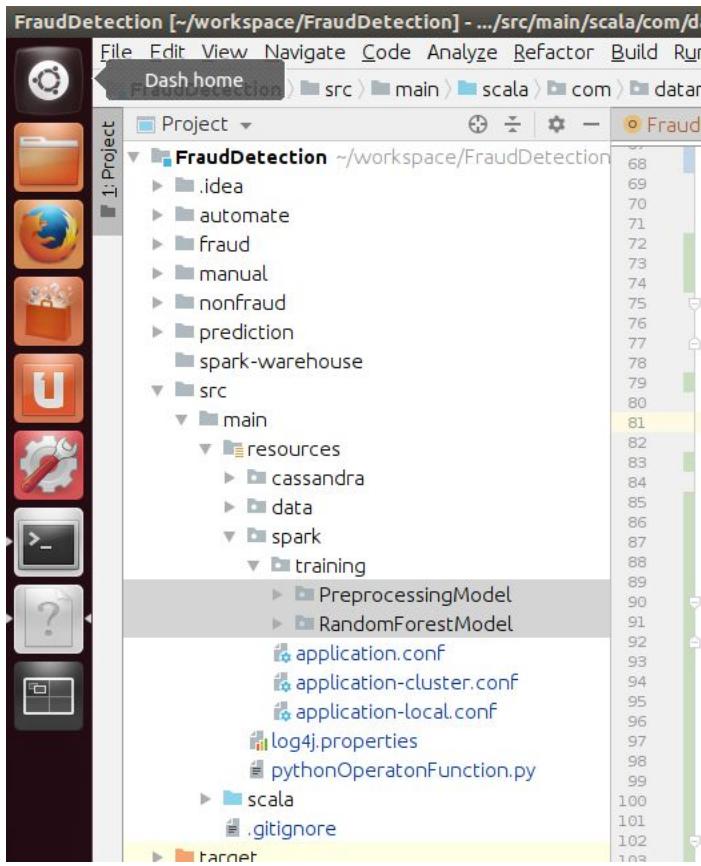
If there are more genuine transactions relative to fraud transactions then it is better to use **Precision** instead of **False Positive Rate**

This seems to be a decent Model. Since our data is a cooked up data, we will not do more evaluation. Usually, different Classification Model must be tried and then the model efficiency must be computed using ROC(**Receiver Operating Characteristics**) curve and finally models from different Algorithm must be compared using AOC(**Area Under Curve**). Once you get a good model, it must be deployed.

5.2.10 Save Model

```
/* Save Preprocessing and Random Forest Model */
randomForestModel.save(SparkConfig.modelPath)
preprocessingTransformerModel.save(SparkConfig.preprocessingModelPath)
```

Both preProcessing Model and Random Forest Model is saved to the file system



5.3 Demonstration

Run FraudDetectionTraining.scala from IntelliJ

No Input Parameters

Right-click on FraudDetectionTraining.scala file and Click Run

```

FraudDetection [-/workspace/FraudDetection] - .../src/main-scala/com/datamana/spark/jobs/FraudDetectionTraining.scala [FraudDetection] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Project FraudDetection -/workspace/FraudDetection
  +--.idea
  +--automate
  +--fraud
  +--manual
    +--customers.csv
    +--sparkFraudDetectionTraining.sh
    +--sparkImportToCassandraJob.sh
    +--sparkRealtimeFraudDetectionStream.sh
    +--sparkRealtimeFraudDetectionStructuredStream.sh
    +--transactions.csv
  +--nonfraud
  +--prediction
  +--spark-warehouse
  +--src
    +--main
      +--resources
      +--scala
        +--com.datamana
          +--cassandra
          +--config
          +--creditcard
          +--kafka
          +--spark
            +--Jobs
              +--RealTimeFraudDetection
                +--FraudDetectionTraining
                +--InitialImportToCassandra
                +--SparkJob
            +--pipeline
            +--DataBalancing
            +--DataReader
            +--GracefulShutdown
    +--TODO
    +--Terminal
    +--Version Control
  +--Structure
  +--Favorites
  +--Event Log
FraudDetectionTraining.scala
transactionDF.cache()

transactionDF.show(truncate = false)

val columnNames = List("category", "merchant", "distance", "amt", "age")

val pipelineStages = BuildPipeline.createFeaturePipeline(transactionDF.schema, columnNames)
val pipeline = new Pipeline().setStages(pipelineStages)
val preprocessingTransformerModel = pipeline.fit(transactionDF)

val featureDF = preprocessingTransformerModel.transform(transactionDF)

featureDF.show(truncate = false)

val Array(trainData, testData) = featureDF.randomSplit(Array(0.8, 0.2))

val featureLabelDF = trainData.select(col("features"), col("label"))

val nonFraudDF = featureLabelDF.filter(condition = $"is_fraud" =:= 0)
val fraudDF = featureLabelDF.filter(condition = $"is_fraud" =:= 1)
val fraudCount = fraudDF.count()

println("FraudCount: " + fraudCount)

// There will be very few fraud transaction and more normal
// transaction so model will not have good prediction accuracy
val balancedNonFraudDF = DataBalancing.createBalancedDataFrame(nonFraudDF, fraudDF)

val finalFeatureDF = fraudDF.union(balancedNonFraudDF)

val randomForestModel = AlgoTrees.randomForestClassifier()
val predictionDF = randomForestModel.transform(testData)
predictionDF.show(truncate = false)

val predictionAndLabels = FraudDetectionTraining > main(args: Array[String])

```

Copy Reference Ctrl+Alt+Shift+C
 Paste Ctrl+V
 Paste from History... Ctrl+Shift+F
 Paste without Formatting Ctrl+Alt+Shift+V
 Column Selection Mode Alt+Shift+Insert
 Refactor
 Folding
 Analyze
 Go To
 Generate... Alt+Insert
 Recompile '...ctionTraining.scala' Ctrl+Shift+F9
Run FraudDetectionTraining Ctrl+Shift+F10
 Debug 'FraudDetectionTraining'
 Run 'FraudDetectionTraining' with Coverage
 Select 'FraudDetectionTraining'
 Show in File Manager
 Open in Terminal
 Run Scala Console Ctrl+Shift+F9
 Local History
 Git
 Compare with Clipboard
 File Encoding
 Open on GitHub
 Create Gist...

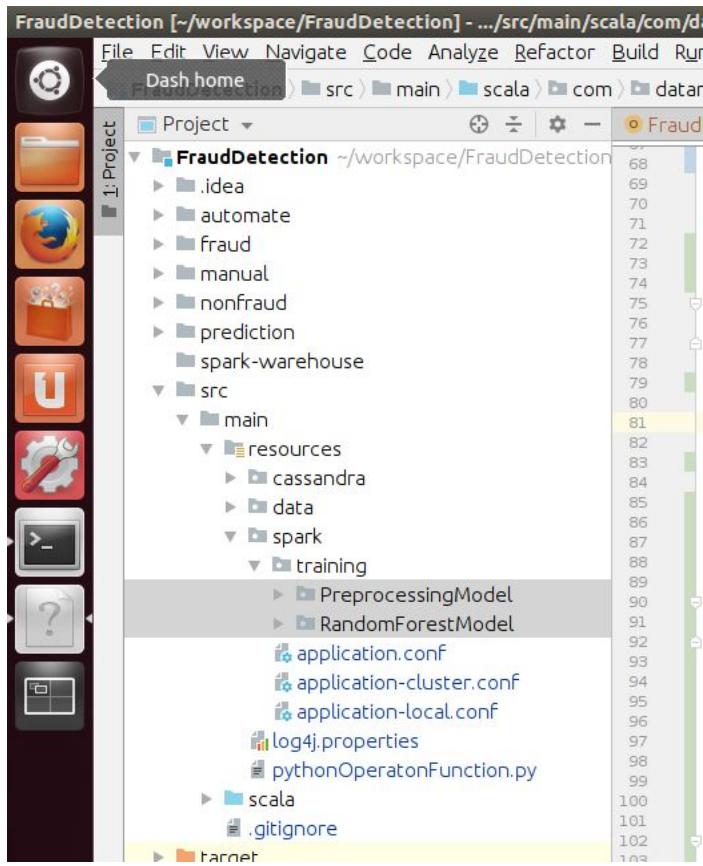
IDE and Plugin Updates
 IntelliJ IDEA is ready to update.

Output

```

===== Confusion matrix =====
#####| Actual = 1           Actual = 0
-----+
Predicted = 1| 91.000000   83.000000
Predicted = 0| 0.000000   2548.000000
-----
True Positive Rate: 1.0
False Positive Rate: 0.03154694
Precision: 0.5229885

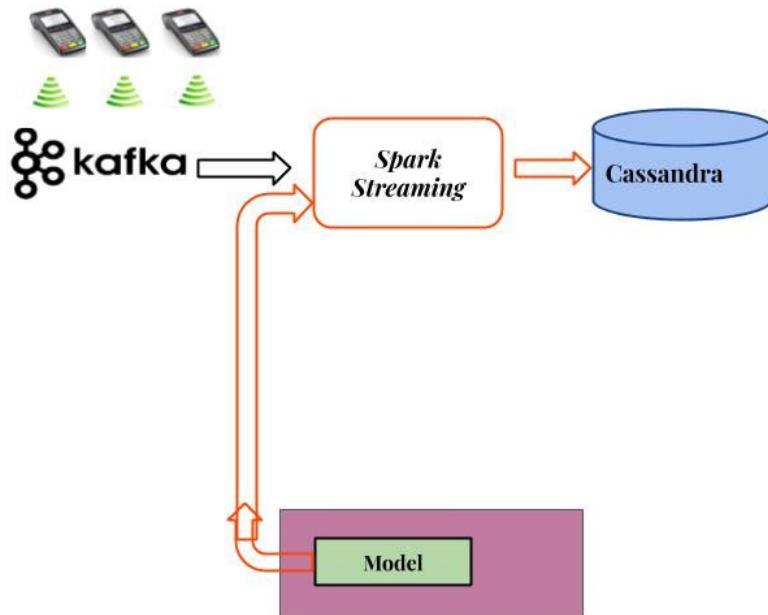
```



Preprocessing Model and Random Forest Model is saved to the File System

6. Spark Streaming

6.1 Introduction



Spark Streaming Job will load both Preprocessing Model and the Random Forest model that was created by the Spark ML Job.

It will start consuming credit card transaction messages from Kafka.

Using Preprocessing Model it will transform all the necessary columns(merchant, category, distance, amt, age) to feature column.

Using Random Forest Model it will predict whether a transaction is a fraud or not.

Predicted transactions will be saved to Cassandra Keyspace

Along with saving predicted transaction to Cassandra, we are also saving Kafka offset to Cassandra

In Fraud Detection, transaction must not be lost nor there must be any duplicate transactions i.e it must be exactly-once semantics. Here we are achieving exactly-once semantics by manually committing offset

6.2 Fraud Prediction

6.2.1 Read Customer data

Read customer data and extract the age column

```
import sparkSession.implicits._

val customerDF = DataReader.readFromCassandra(CassandraConfig.keyspace,
CassandraConfig.customer)

val customerAgeDF = customerDF.withColumn("age",
(datediff(current_date(), to_date($"dob"))/365).cast(IntegerType))

customerAgeDF.cache()
```

6.2.2 Load Models

Load Preprocessing Model and Random Forest Model from the previous Spark ML Job

```
val preprocessingModel =
PipelineModel.load(SparkConfig.preprocessingModelPath)

val randomForestModel =
RandomForestClassificationModel.load(SparkConfig.modelPath)
```

6.2.3 Read Offset

As soon as the streaming is started. First, we need to get the offset. Using this offset we can fetch messages from Kafka. We are saving offset to Cassandra. Hence reading it from Cassandra.

```
val storedOffsets = CassandraDriver.readOffset(CassandraConfig.keyspace,
                                                CassandraConfig.kafkaOffsetTable,
                                                KafkaConfig.kafkaParams("topic"))
```

6.2.4 Fetch Messages from Kafka

```
val ssc = new StreamingContext(sparkSession.sparkContext,
Duration(SparkConfig.batchInterval))
```

```

val stream = storedOffsets match {
  case None => {
    KafkaUtils.createDirectStream[String, String](ssc,
      PreferConsistent,
      Subscribe[String, String](topics, kafkaParams)
    )
  }

  case Some(fromOffsets) => {
    KafkaUtils.createDirectStream[String, String](ssc,
      PreferConsistent,
      Assign[String, String](fromOffsets.keys.toList,
      kafkaParams, fromOffsets))
  }
}

```

First we are creating Streaming Context.

Next, we are fetching messages from Kafka using Direct Stream.

For the very first time, there will be no offset in Cassandra. Hence `storedOffsets` will be `None`. So messages will be fetched from the starting offset.

Let's say after some time Spark Streaming Job got restarted due to some error. By this time we have saved some offset to Cassandra. `storedOffsets` will be `Some`. Hence messages will be fetched from where the Streaming was restarted and not from the beginning

6.2.5 Prediction

```

transactionStream.foreachRDD(rdd => {

  if (!rdd.isEmpty()) {
    .....
    val featureTransactionDF =
preprocessingModel.transform(processedTransactionDF)
    val predictionDF =
randomForestModel.transform(featureTransactionDF)
      .withColumnRenamed("prediction", "is_fraud")
    .....
  }
}

```

On the received messages, Preprocessing Model is applied to transform columns into

the feature column. Then Random Forest Model is applied to predict whether a transaction is fraud or not.

6.3 Save Prediction

```
val preparedStatementFraud =  
    session.prepare(CreditcardTransactionRepository.cqlTransactionPrepare(keyspace,  
        fraudTable))  
    val preparedStatementNonFraud =  
        session.prepare(CreditcardTransactionRepository.cqlTransactionPrepare(keyspace,  
            nonFraudTable))  
  
partitionOfRecords.foreach(record => {  
    val isFraud = record.getAs[Double]("is_fraud")  
    if (isFraud == 1.0) {  
        // Bind and execute prepared statement for Fraud Table  
  
        session.execute(CreditcardTransactionRepository.cqlTransactionBind(prepared  
            StatementFraud, record))  
    }  
    else if(isFraud == 0.0) {  
        // Bind and execute prepared statement for NonFraud Table  
  
        session.execute(CreditcardTransactionRepository.cqlTransactionBind(prepared  
            StatementNonFraud, record))  
    }  
    .....  
})
```

If the transaction is fraud it is saved to fraud_transacation table

If the transaction is not fraud it is saved to non_fraud_transaction table

6.4 Save Offset

Get the max offset from the current batch and save it to kafka_offset Cassandra table

```
val preparedStatementOffset =  
    session.prepare(KafkaOffsetRepository.cqlOffsetPrepare(keyspace,  
        kafkaOffsetTable))
```

```

val partitionOffset:Map[Int, Long] = Map.empty
partitionOfRecords.foreach(record => {
    .....
    //Get max offset in the current match
    val kafkaPartition = record.getAs[Int]("partition")
    val offset = record.getAs[Long]("offset")
    partitionOffset.get(kafkaPartition) match {
        case None => partitionOffset.put(kafkaPartition, offset)
        case Some(currentMaxOffset) =>
            if(offset > currentMaxOffset)
                partitionOffset.update(kafkaPartition, offset)
    }
}
partitionOffset.foreach(t => {
    // Bind and execute prepared statement for Offset Table
    session.execute(KafkaOffsetRepository.cqlOffsetBind(preparedStatementOffset
, t))
})

```

6.5 Exactly-once Semantics

Whenever the Streaming Job restarts due to some error neither a single transaction must be lost nor there must be any duplicate transactions. This is exactly-once semantics

From Section 6.3 and 6.4, we can see that first, we are saving transactions to transaction tables. Next, we are saving offset to the `kafka_offset` table. Suppose after saving transactions, our Streaming job stopped due to some error. So only transactions of the current batch are saved and the offset is not saved. As soon as it is restarted, Streaming Job will fetch messages(transactions) from the previous offset. So now we have duplicate transactions. These duplicate transactions are saved to Cassandra. Transaction tables in Cassandra are designed to be idempotent writes. We have chosen `(cc_num, transaction_time)` as the primary key. So if we get duplicate transaction it will be overwritten as there cannot be duplicate rows for the same primary key.

This is how we are achieving exactly-once semantics. This is one of the ways to achieve exactly-once semantics.

6.6 Shutdown Spark Streaming Job

This function is used to Programmatically shutdown Spark Streaming Job. Whenever a new model is created we have to stop the currently running Streaming Job and must start a new Streaming Job which will pick the new model. A thread in Spark Streaming Job will call this function. This function will check if a dummy file, `/tmp/shutdownmarker`, exist or not. If it existing then this function will stop the Streaming Job.

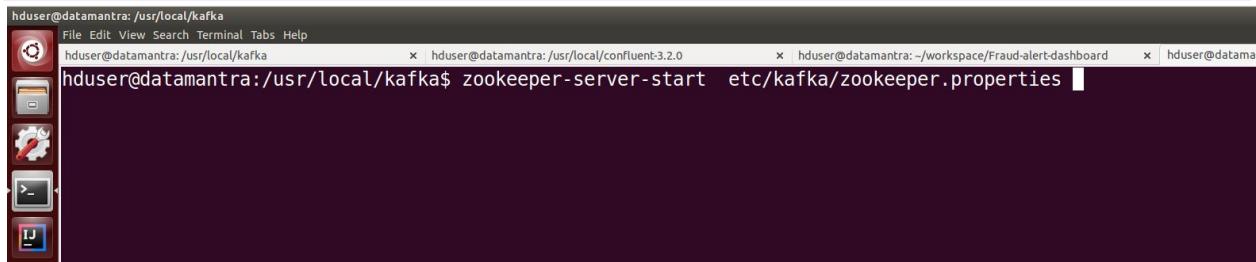
This dummy file is created in the automation script([fraudetection.py](#))

```
def handleGracefulShutdown(checkIntervalMillis:Int,  
    ssc:StreamingContext)(implicit sparkSession: SparkSession) {  
  
    var isStopped = false  
  
    while (! isStopped) {  
        logger.info("calling awaitTerminationOrTimeout")  
        isStopped = ssc.awaitTerminationOrTimeout(checkIntervalMillis)  
        if (isStopped)  
            logger.info("confirmed! The streaming context is stopped. Exiting  
application...")  
        else  
            logger.info("Streaming App is still running. Timeout...")  
        checkShutdownMarker  
        if (!isStopped && stopFlag) {  
            logger.info("stopping ssc right now")  
            ssc.stop(true, true)  
            sparkSession.stop  
            logger.info("ssc is stopped!!!!!!")  
        }  
    }  
}
```

6.7 Demonstration

Start Zookeeper

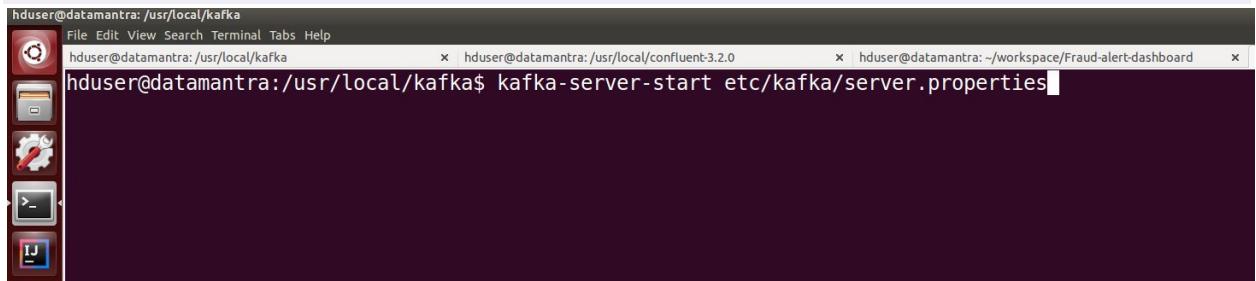
```
cd /usr/local/kafka  
zookeeper-server-start etc/kafka/zookeeper.properties
```



A screenshot of a terminal window titled "hduser@datamantra:/usr/local/kafka". The window contains three tabs: "hduser@datamantra:/usr/local/kafka", "hduser@datamantra:/usr/local/confluent-3.2.0", and "hduser@datamantra:~/workspace/Fraud-alert-dashboard". The current tab shows the command "zookeeper-server-start etc/kafka/zookeeper.properties" being typed.

Start Kafka Server

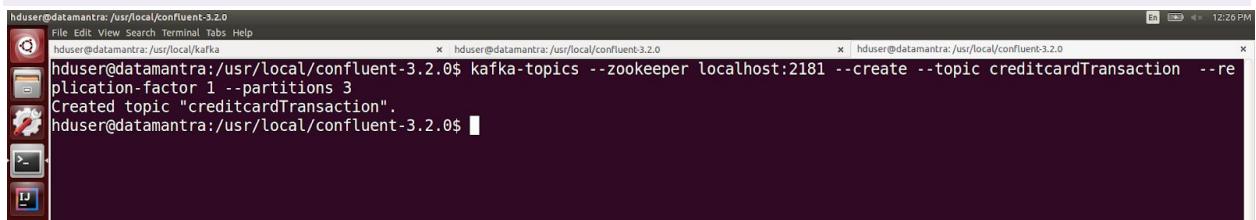
```
kafka-server-start etc/kafka/server.properties
```



A screenshot of a terminal window titled "hduser@datamantra:/usr/local/kafka". The window contains three tabs: "hduser@datamantra:/usr/local/kafka", "hduser@datamantra:/usr/local/confluent-3.2.0", and "hduser@datamantra:~/workspace/Fraud-alert-dashboard". The current tab shows the command "kafka-server-start etc/kafka/server.properties" being typed.

Create transaction topic

```
kafka-topics --zookeeper localhost:2181 --create --topic  
creditcardTransaction --replication-factor 1 --partitions 3
```

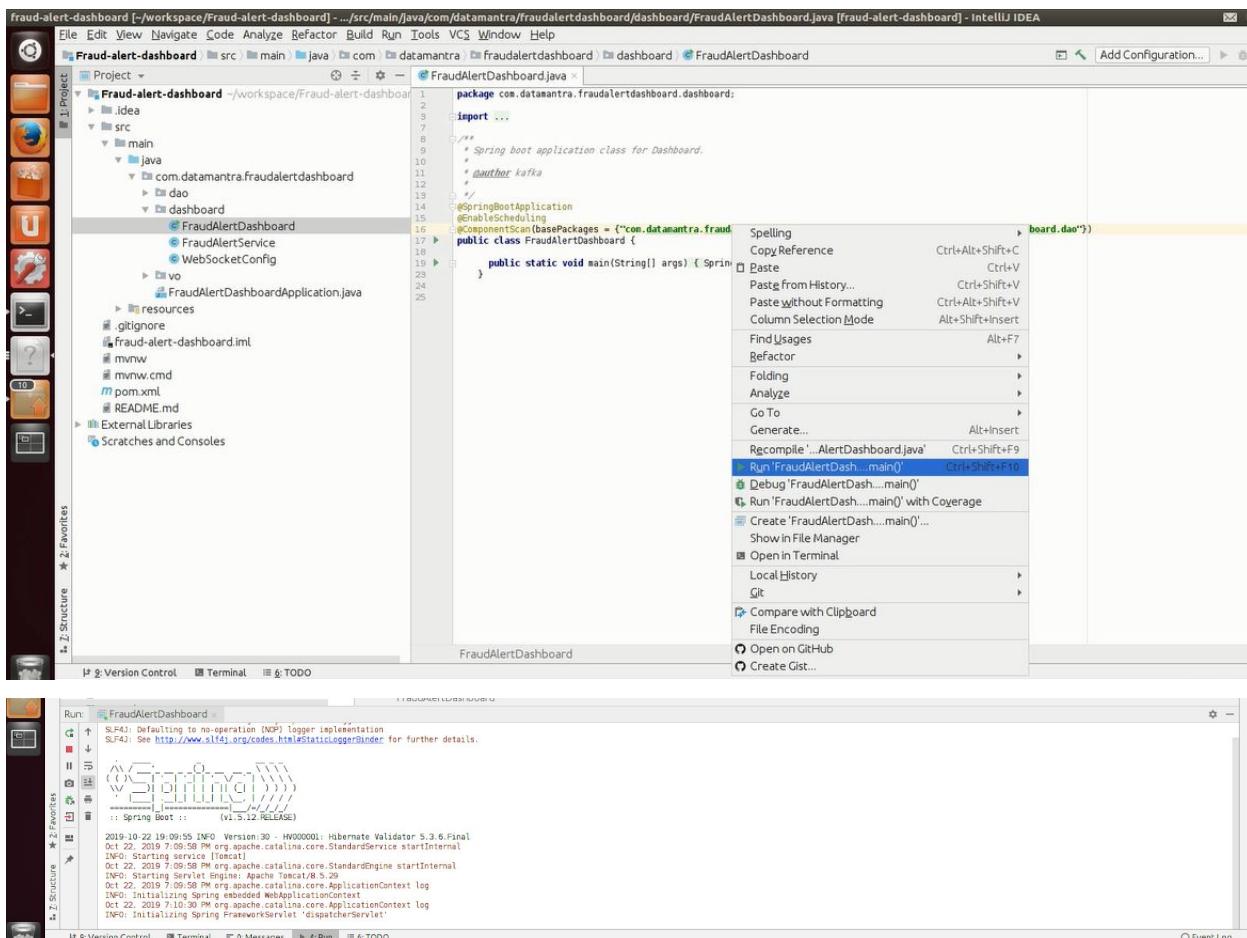


A screenshot of a terminal window titled "hduser@datamantra:/usr/local/confluent-3.2.0". The window contains three tabs: "hduser@datamantra:/usr/local/kafka", "hduser@datamantra:/usr/local/confluent-3.2.0", and "hduser@datamantra:~/workspace/Fraud-alert-dashboard". The current tab shows the command "kafka-topics --zookeeper localhost:2181 --create --topic creditcardTransaction --replication-factor 1 --partitions 3" being run, with the output "Created topic \"creditcardTransaction\"." displayed.

Run Fraud-alert-dashboard project from IntelliJ

No input parameters

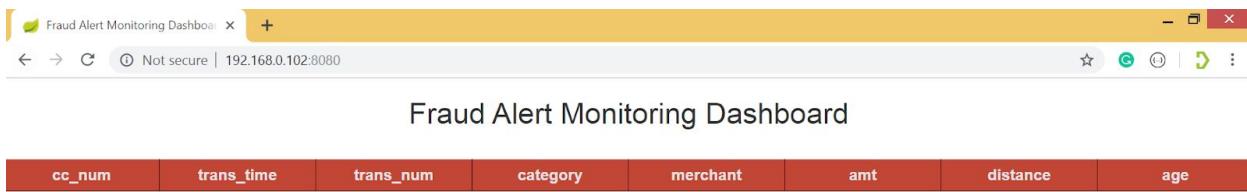
Right Click on FraudAlertDashboard.java file and Run



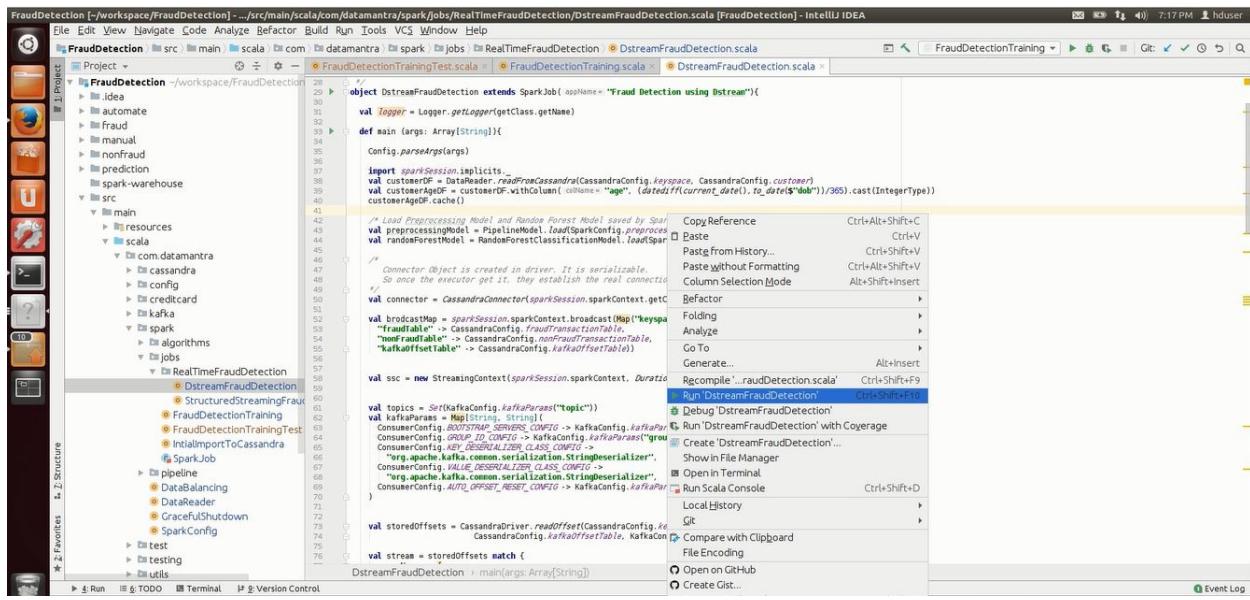
Output

The fraud alert dashboard web server is up and running. Verify this by accessing through the web browser

<http://<ubuntu-ipadd>:8080>



Run Spark Streaming Job from IntelliJ



The screenshot shows the IntelliJ IDEA interface with the project navigation bar at the top. Below it is the code editor with a Scala file named `DstreamFraudDetection.scala`. A context menu is open over the code, specifically over the line `val ssc = new StreamingContext(sparkSession.sparkContext, Duration`. The menu items visible include:

- Copy Reference
- Paste
- Paste from History...
- Paste without Formatting
- Column Selection Mode
- Refactor
- Folding
- Analyze
- Go To
- Generate...
- Recompile '...raudDetection.scala'
- Run 'DstreamFraudDetection'
- Debug 'DstreamFraudDetection'
- Run 'DstreamFraudDetection' with Coverage
- Create 'DstreamFraudDetection'
- Show in File Manager
- Open in Terminal
- Run Scala Console
- Local History
- Git
- Compare with Clipboard
- File Encoding
- Open on GitHub
- Create Gist...

Below the code editor is the run tool window, which shows the output of the application. The log output is as follows:

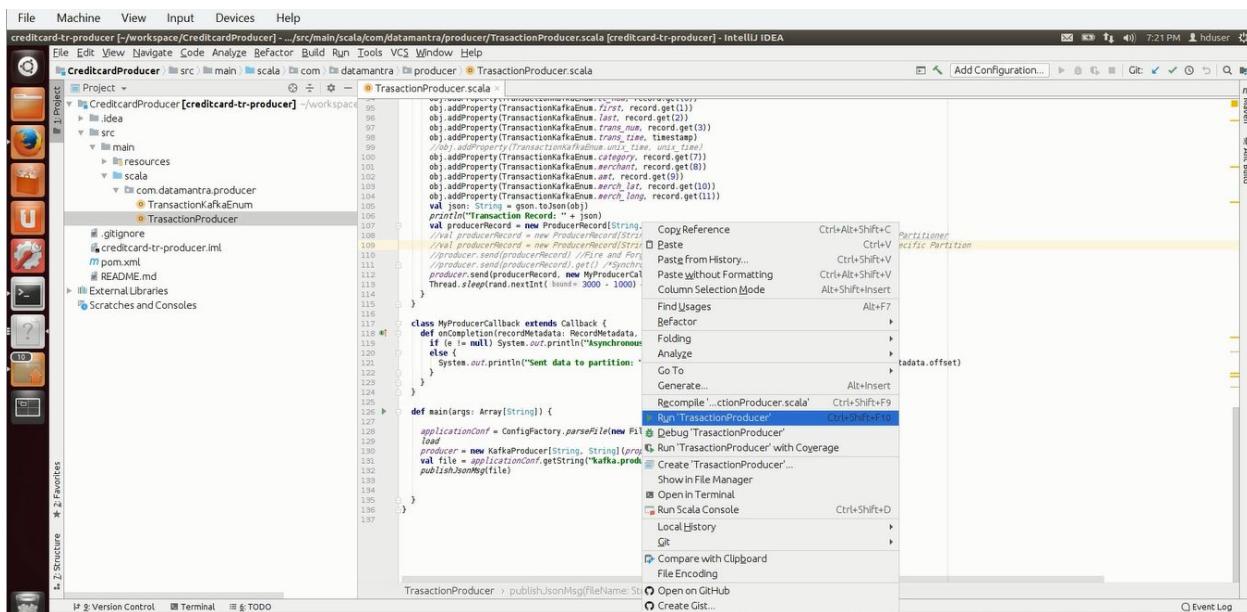
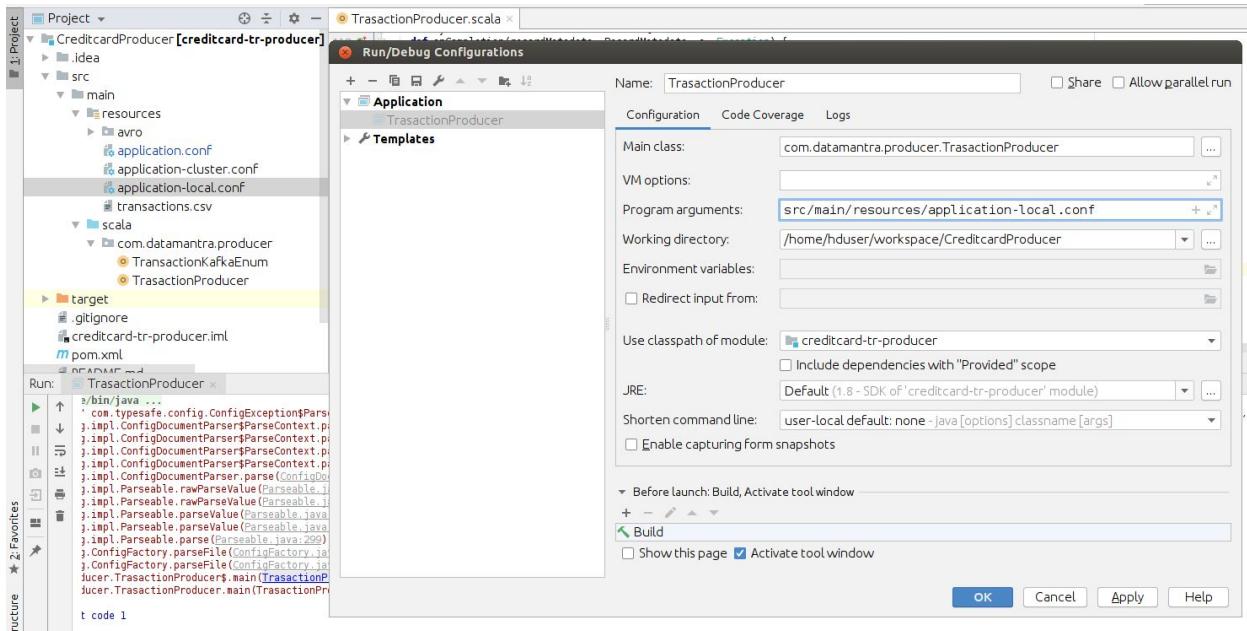
```
19/10/22 19:20:16 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:16 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:17 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:17 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:18 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:18 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:19 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:19 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:20 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:20 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:21 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:21 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:22 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:22 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:23 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:23 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:24 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:24 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:25 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:25 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
19/10/22 19:20:26 INFO com.datamana.spark GracefulShutdown: Streaming App is still running. Timeout...
19/10/22 19:20:26 INFO com.datamana.spark GracefulShutdown: calling awaitTerminationOrTimeout...
```

Run CreditcardProducer from IntelliJ

<https://github.com/pramoddatamantra/CreditcardProducer>

Input parameter is the configuration file

```
src/main/resources/application-local.conf
```



Output

Whenever Spark Streaming Job predicts a fraud transaction it will be displayed on Fraud Alert Dashboard

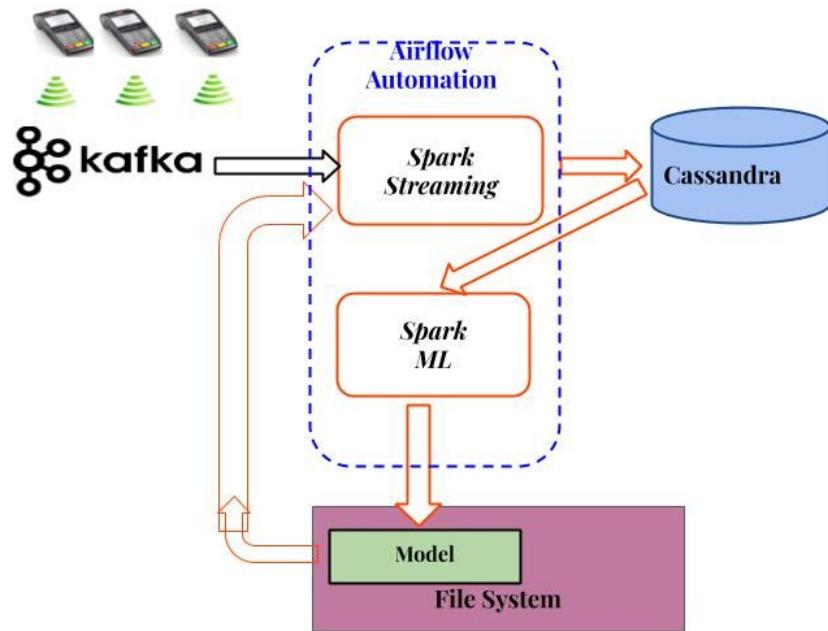


Fraud Alert Monitoring Dashboard

cc_num	trans_time	trans_num	category	merchant	amt	distance	age
374115112731710	2019-10-22 19:30:50	11efb54624f40fe0...	entertainment	Johns-Hoeger	64	4.16	41

7. Airflow Automation

7.1 Introduction



Apache Airflow is an automation framework. It is used to automate Jobs. It is similar to oozie. You can write workflows using Python language instead of XML workflows. It is much easier than oozie.

Here we are automating both Spark Streaming and Spark ML Job. But in reality, Spark ML Job is not automated. It is manually run to create a Model. The model will be evaluated for efficiency. If the Model is efficient, only then it will be deployed. But here we will be automating Spark ML Job as well.

As new transaction data is captured, we need to recreate the model using both old data and new data. So the new model will be created usually once a week or once a month. But for the sake of demonstration, we run Spark ML Job every 20 mins. Once the new model is created, Spark Streaming Job has to pick this new Model. So we will stop the currently running Spark Streaming Job and start a new Streaming Job. This new Streaming Job will load the new Model and will start the prediction. So Airflow automation is used to create a new model every 20 mins and to Stop and Start Spark Streaming Job whenever a new model is created.

7.2 Automation Steps

1. Remove Old Model

This training folder contains Preprocessing Model and Random Forest Model. We are just deleting the folder

```
remove_model = BashOperator(  
    task_id='remove_model',  
    bash_command='rm -rf' + ' ' + home +  
'/frauddetection/spark/training',  
    default_args=default_args)
```

2. Create New Model

Here we are creating a new model by running Spark ML Job through spark-submit command

```
create_model = BashOperator(  
    task_id='create_model',  
    bash_command='spark-submit --class  
com.datamantra.spark.jobs.FraudDetectionTraining --name "Fraud Detection  
Spark ML Training" --master spark://datamantra:7077 --total-executor-cores  
1' + ' ' + home + '/frauddetection/spark/frauddetection-spark.jar' + ' ' +  
home + '/frauddetection/spark/application-local.conf',  
    default_args=default_args)
```

3. Sleep for 5 seconds(This step is optional)

```
sleep = BashOperator(  
    task_id='sleep',  
    bash_command='sleep 5')
```

4. Stop and Start Spark Streaming Job.

Here we are call `stopStartStreamingJob` function.

```
stop_start_streaming = PythonOperator(  
    task_id='stop_start_streaming',  
    python_callable=stopStartStreamingJob)
```

In this function, first, we are stopping the currently running Streaming Job by creating a dummy file `/tmp/shutdownmarker`. A thread running in the Streaming Job will be

continuously checking for this file. As soon as this file is created, Streaming Job will be shutdown.

Next, we are waiting for the Streaming job to shutdown. As soon as it is shut down, we are deleting the dummy file so that newly started Streaming job must not get shutdown. Finally, we are starting the new Streaming job using the spark-submit command

```
def stopStartStreamingJob():
    stop_streaming = 'touch /tmp/shutdownmarker'
    os.system(stop_streaming)
    shutdown_flag = False

    while not shutdown_flag:
        URL = "http://localhost:8080/json"
        r = requests.get(url = URL)
        data = json.loads(r.content)
        activeapps = data['activeapps']
        if not activeapps:
            logger.info("No Active apps, Streaming app is shutdown")
            shutdown_flag = True
        else:
            logger.info("List is not empty")
            activeFlag = False
            for app in activeapps:
                if app['name'] == 'RealTime Creditcard FraudDetection':
                    logger.info("Streaming Job is still running")
                    activeFlag = True
                    break
            if activeFlag == False:
                logger.info("Streaming Job not in activeapps list.")
                shutdown_flag = True
            else:
                logger.info("Streaming Job is still Running")

    remove_shutdown_marker = 'rm -rf /tmp/shutdownmarker'
    os.system(remove_shutdown_marker)

    start_streaming = 'spark-submit --class
com.datamantra.spark.jobs.RealTimeFraudDetection.DstreamFraudDetection
--name "RealTime Creditcard FraudDetection" --master
spark://datamantra:6066 --deploy-mode cluster --total-executor-cores 1' +
' ' + home + '/frauddetection/spark/fraudetection-spark.jar' + ' ' + home
+ '/frauddetection/spark/application-local.conf'
```

```
os.system(start_streaming)
```

7.3 Configure Airflow

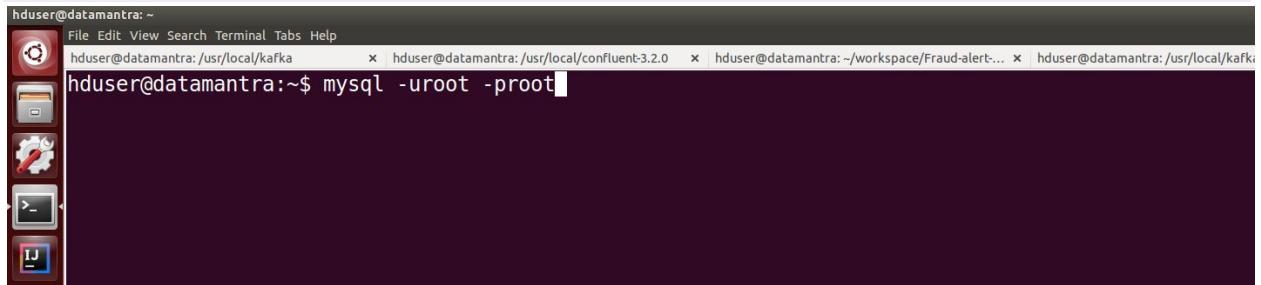
By default, apache airflow is configured with SQLite. But SQLite is not suitable for the Production environment. Hence modify configure airflow to use MySQL database

7.3.1 Create hduser and airflow database

Create a new user and database in Mysql

Login to Mysql with root credentials

```
mysql -uroot -proot
```



Create hduser with hadoop123 as password

```
CREATE USER 'hduser'@'localhost' IDENTIFIED BY 'hadoop123';
GRANT ALL PRIVILEGES ON * . * TO 'hduser'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

```
hduser@pixipanda:~/idea-IC-183.6156.11/bin * hduser@pixipanda:~/idea-IC-183.6156.11/bin * hduser@pixipanda:~/idea-IC-183.6156.11/bin * hduser@pixipanda:~/idea-IC-183.6156.11/bin
mysql>
mysql> CREATE USER 'hduser'@'localhost' IDENTIFIED BY 'hadoop123';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON * . * TO 'hduser'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> EXIT;
Bye
hduser@pixipanda:~$
```

Login as hduser

```
mysql -uhduser -phadoop123
```

```
hduser@datamantra:~ File Edit View Terminal Tabs Help
hduser@datamantra:/usr/local/kafka * hduser@datamantra:/usr/local/confluent-3.2.0 * hduser@datamantra:~/workspace/Fraud-alert... * hduser@datamantra:/usr/local/kafka
hduser@datamantra:~$ mysql -uhduser -phadoop123
```

Create Airflow database and exit

```
CREATE DATABASE airflow;
EXIT;
```

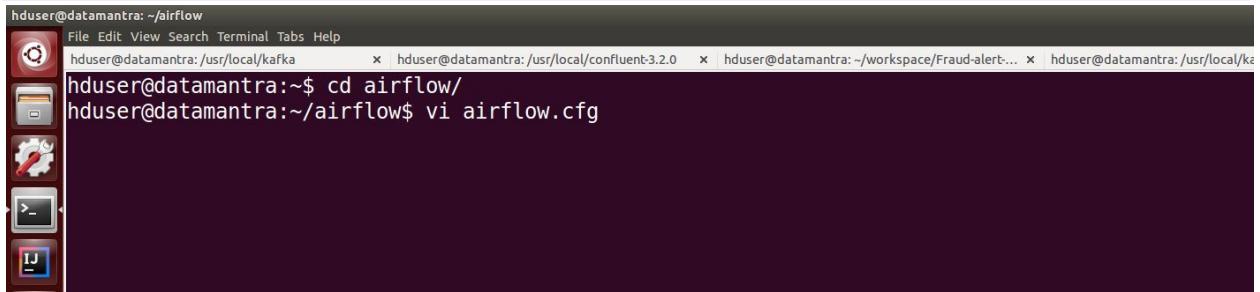
```
hduser@pixipanda:~
hduser@pixipanda:~/idea-IC-183.6156.11/bin * hduser@pixipanda:~/idea-IC-183.6156.11/bin * hduser@pixipanda:~/idea-IC-183.6156.11/bin * hduser@pixipanda:~/idea-IC-183.6156.11/bin
mysql> CREATE DATABASE airflow;
Query OK, 1 row affected (0.00 sec)

mysql> exit;
Bye
hduser@pixipanda:~$
```

7.3.2 Modify airflow.cfg

Modify the airflow.cfg file to use MySQL database

```
cd ~/airflow  
vi airflow.cfg
```



```
hduser@datamantra: ~/airflow  
File Edit View Search Terminal Tabs Help  
hduser@datamantra:/usr/local/kafka x hduser@datamantra:/usr/local/confluent-3.2.0 x hduser@datamantra:~/workspace/Fraud-alert... x hduser@datamantra:/usr/local/ke...  
hduser@datamantra:~$ cd airflow/  
hduser@datamantra:~/airflow$ vi airflow.cfg
```

Modify following fields, save and exit

```
sql_alchemy_conn = mysql://hduser:hadoop123@localhost:3306/airflow  
load_examples = False
```

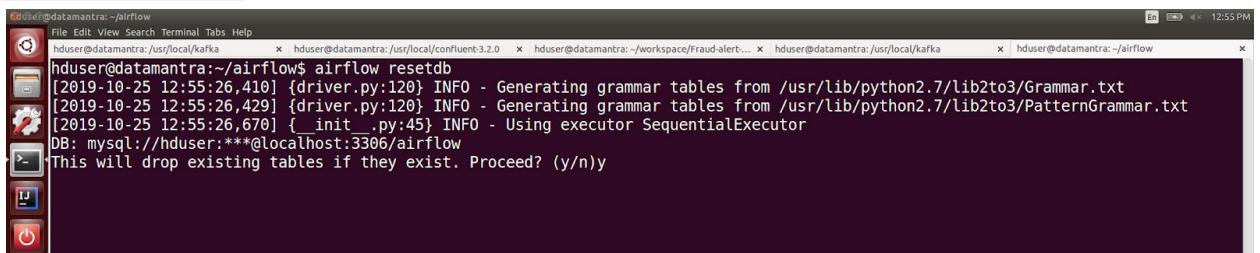
Here we are configuring MySQL instead of SQLite.

Also instead of loading all the built-in example scripts of Airflow, only our script will be loaded.

```
# The SQLAlchemy connection string to the metadata database.  
# SQLAlchemy supports many different database engine, more information  
# their website  
#sql_alchemy_conn = sqlite:///home/hduser/airflow/airflow.db  
sql_alchemy_conn = mysql://hduser:hadoop123@localhost:3306/airflow  
  
# Whether to load the examples that ship with Airflow. It's good to  
# get started, but you probably want to set this to False in a production  
# environment  
load_examples = False
```

Rest Airflow Database

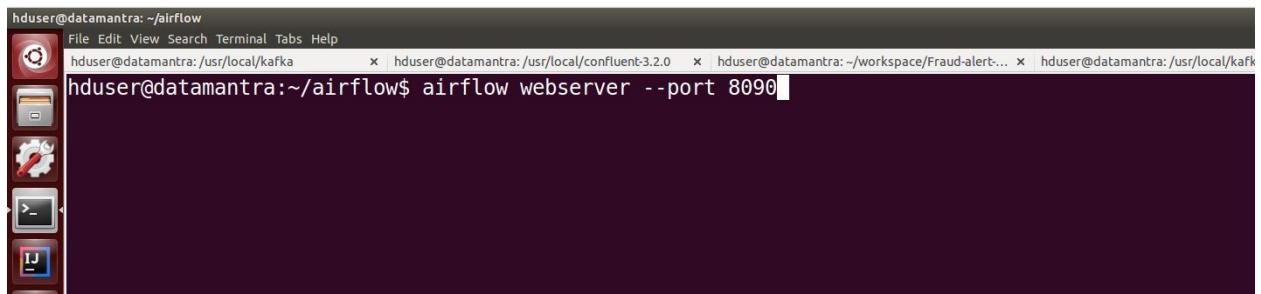
```
airflow resetdb
```



```
hduser@datamantra: ~/airflow  
File Edit View Search Terminal Tabs Help  
hduser@datamantra:/usr/local/kafka x hduser@datamantra:/usr/local/confluent-3.2.0 x hduser@datamantra:~/workspace/Fraud-alert... x hduser@datamantra:/usr/local/ke...  
hduser@datamantra:~/airflow$ airflow resetdb  
[2019-10-25 12:55:26,410] {driver.py:120} INFO - Generating grammar tables from /usr/lib/python2.7/lib2to3/Grammar.txt  
[2019-10-25 12:55:26,429] {driver.py:120} INFO - Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt  
[2019-10-25 12:55:26,670] {__init__.py:45} INFO - Using executor SequentialExecutor  
DB: mysql://hduser:***@localhost:3306/airflow  
This will drop existing tables if they exist. Proceed? (y/n)
```

Start Airflow Webserver

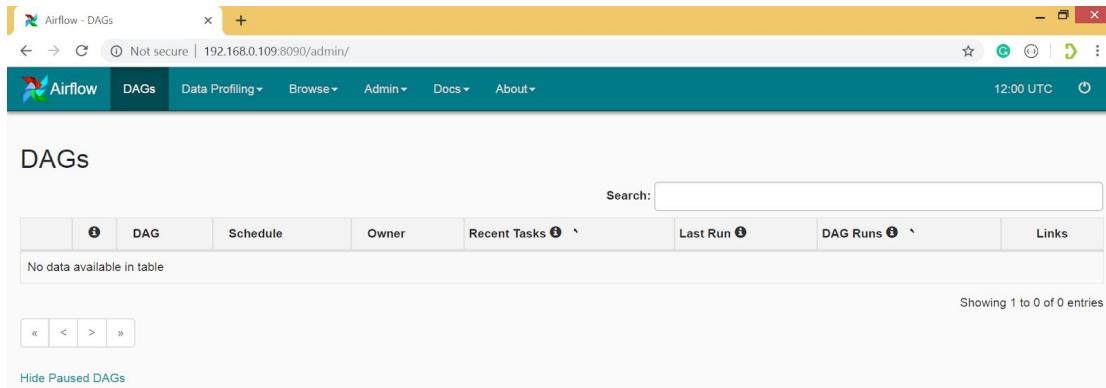
```
airflow webserver --port 8090
```



A terminal window titled 'hduser@datamantra: ~/airflow' with the command 'airflow webserver --port 8090' entered and running.

Access Airflow Web Server

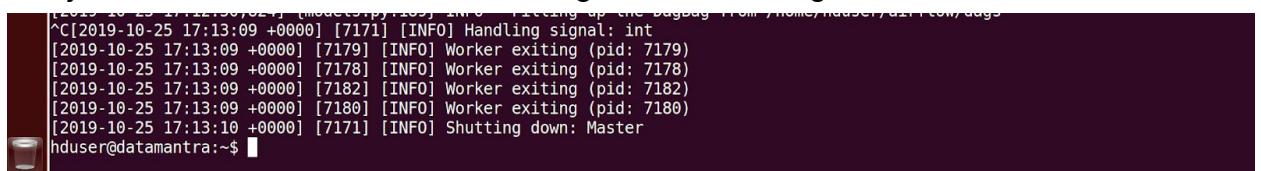
Access https://<ubunt_ipaddress>:8080 to verify whether the airflow web server is up and running



Airflow is configured successfully

Stop Airflow Webserver

Stop Airflow Webserver by pressing CTRL C on the terminal where it was started. This was just to test whether Airflow is working. We will start again when we start automation



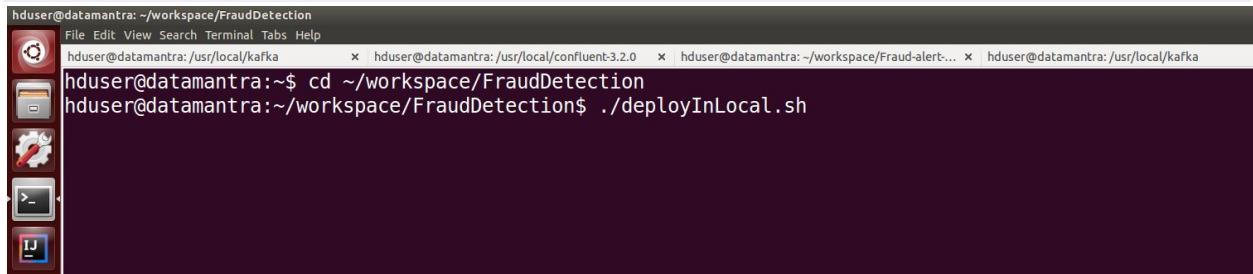
A terminal window showing the airflow webserver stopping. The logs show multiple workers exiting and the master shutting down. The command 'ctrl+c' is shown at the end.

```
[2019-10-25 17:13:08,027] {__init__.py:105} INFO - Setting up the dagbag from /home/hduser/airflow/dags
^[[2019-10-25 17:13:09 +0000] [7171] [INFO] Handling signal: int
[2019-10-25 17:13:09 +0000] [7179] [INFO] Worker exiting (pid: 7179)
[2019-10-25 17:13:09 +0000] [7178] [INFO] Worker exiting (pid: 7178)
[2019-10-25 17:13:09 +0000] [7182] [INFO] Worker exiting (pid: 7182)
[2019-10-25 17:13:09 +0000] [7180] [INFO] Worker exiting (pid: 7180)
[2019-10-25 17:13:10 +0000] [7171] [INFO] Shutting down: Master
hduser@datamantra:~$
```

7.4 Build all Projects

Build FraudDetectoin Project

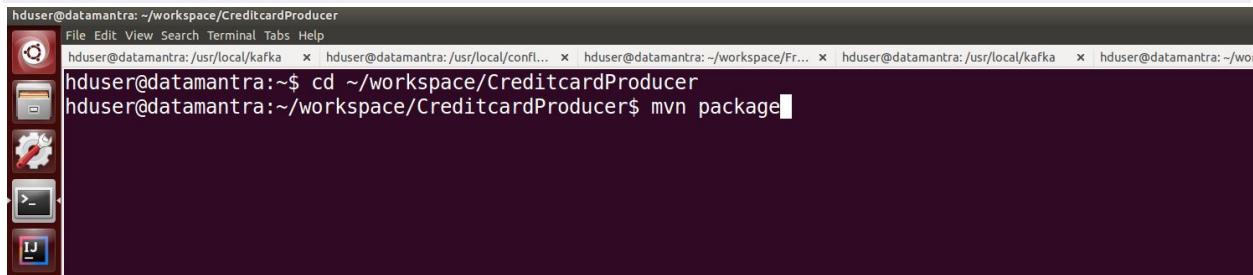
```
cd ~/workspace/FraudDetection  
./deployInLocal.sh
```



A terminal window titled "Terminal" with multiple tabs open. The current tab shows the command "hduser@datamantra:~/workspace/FraudDetection\$./deployInLocal.sh" being run. The output shows the command being executed successfully.

Build Creditcard Producer Project

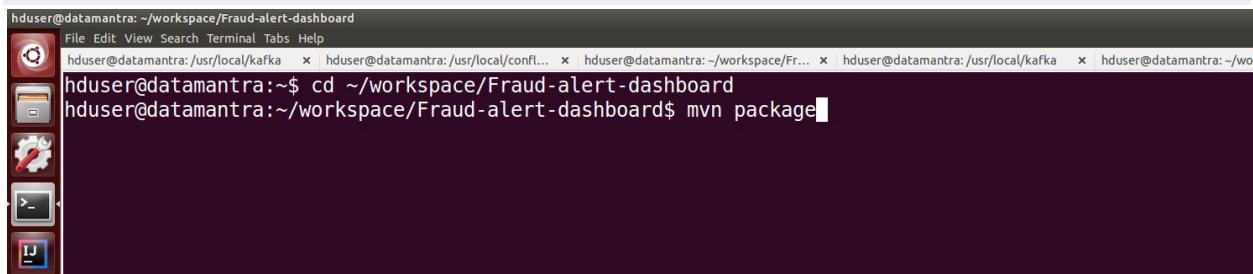
```
cd ~/workspace/CreditcardProducer  
mvn package
```



A terminal window titled "Terminal" with multiple tabs open. The current tab shows the command "hduser@datamantra:~/workspace/CreditcardProducer\$ mvn package" being run. The output shows the command being executed successfully.

Build Fraud-Alert-Dashboard Project

```
cd ~/workspace/Fraud-alert-dashboard  
mvn package
```

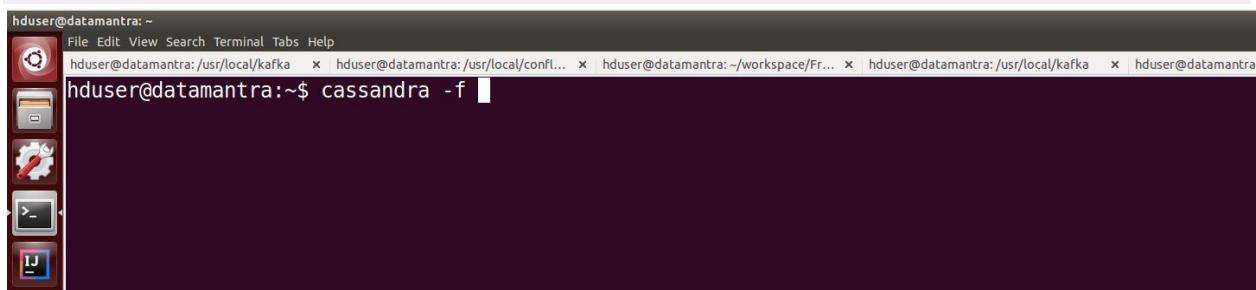


A terminal window titled "Terminal" with multiple tabs open. The current tab shows the command "hduser@datamantra:~/workspace/Fraud-alert-dashboard\$ mvn package" being run. The output shows the command being executed successfully.

7.5 Start all the Servers

Start Cassandra Server

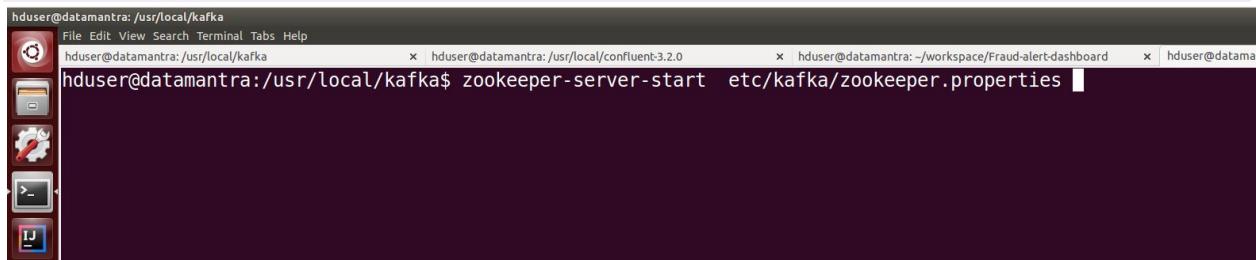
```
cassandra -f
```



A screenshot of a terminal window titled 'hduser@datamantra: ~'. The window shows a command-line interface with the following text:
File Edit View Search Terminal Tabs Help
hduser@datamantra:/usr/local/kafka x hduser@datamantra:/usr/local/confluent... x hduser@datamantra:~/workspace/Fr... x hduser@datamantra:/usr/local/kafka x | hduser@datamantra
hduser@datamantra:~\$ cassandra -f

Start Zookeeper Server

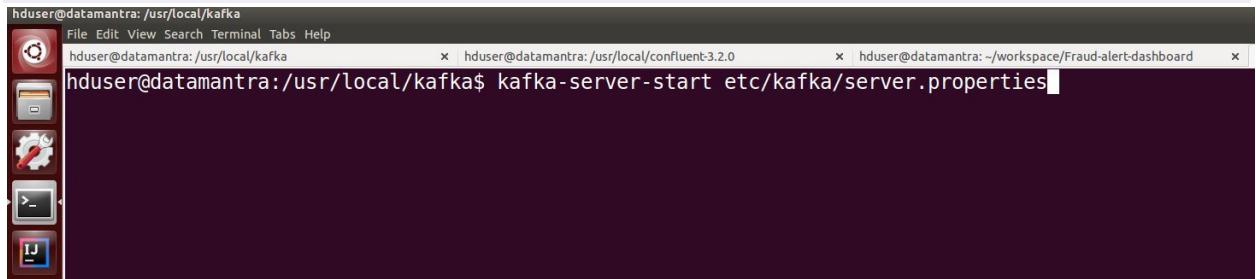
```
cd /usr/local/kafka  
zookeeper-server-start etc/kafka/zookeeper.properties
```



A screenshot of a terminal window titled 'hduser@datamantra:/usr/local/kafka'. The window shows a command-line interface with the following text:
File Edit View Search Terminal Tabs Help
hduser@datamantra:/usr/local/kafka x hduser@datamantra:/usr/local/confluent-3.2.0 x hduser@datamantra:~/workspace/Fraud-alert-dashboard x | hduser@datamantra
hduser@datamantra:/usr/local/kafka\$ zookeeper-server-start etc/kafka/zookeeper.properties

Start Kafka Server

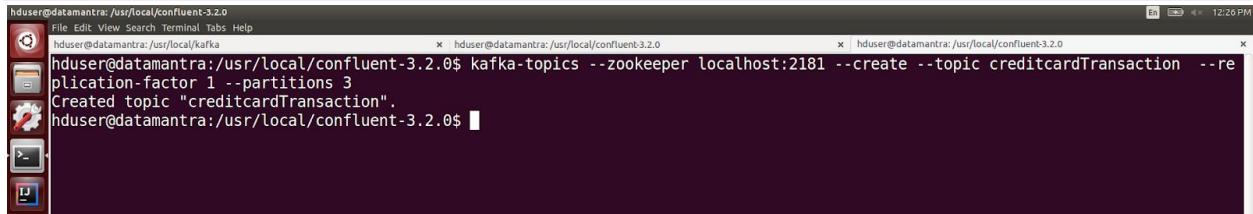
```
kafka-server-start etc/kafka/server.properties
```



A screenshot of a terminal window titled 'hduser@datamantra:/usr/local/kafka'. The window shows a command-line interface with the following text:
File Edit View Search Terminal Tabs Help
hduser@datamantra:/usr/local/kafka x hduser@datamantra:/usr/local/confluent-3.2.0 x hduser@datamantra:~/workspace/Fraud-alert-dashboard x | hduser@datamantra
hduser@datamantra:/usr/local/kafka\$ kafka-server-start etc/kafka/server.properties

Create transaction topic

```
kafka-topics --zookeeper localhost:2181 --create --topic creditcardTransaction --replication-factor 1 --partitions 3
```

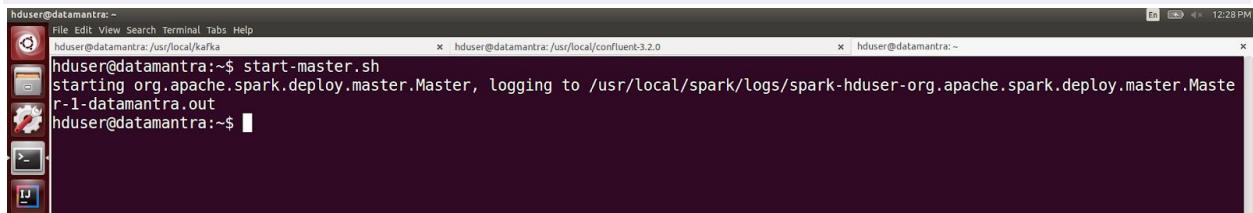


```
hduser@datamantra:/usr/local/confluent-3.2.0
File Edit View Search Terminal Tabs Help
hduser@datamantra:/usr/local/kafka
x | hduser@datamantra:/usr/local/confluent-3.2.0
hduser@datamantra:/usr/local/confluent-3.2.0$ kafka-topics --zookeeper localhost:2181 --create --topic creditcardTransaction --re
plication-factor 1 --partitions 3
Created topic "creditcardTransaction".
hduser@datamantra:/usr/local/confluent-3.2.0$
```

Spark Cluster Setup

Start Spark Master

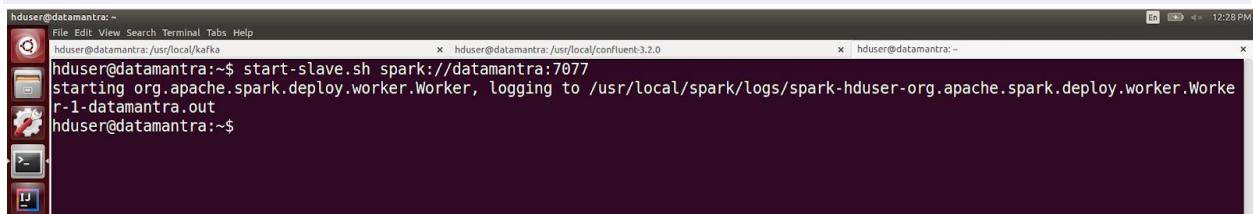
```
start-master.sh
```



```
hduser@datamantra:-
File Edit View Search Terminal Tabs Help
hduser@datamantra:/usr/local/kafka
x | hduser@datamantra:/usr/local/confluent-3.2.0
x | hduser@datamantra:-
hduser@datamantra:~$ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs/spark-hduser-org.apache.spark.deploy.master.Master-1-datamantra.out
hduser@datamantra:~$
```

Start Spark Slave

```
start-slave.sh spark://datamantra:7077
```



```
hduser@datamantra:-
File Edit View Search Terminal Tabs Help
hduser@datamantra:/usr/local/kafka
x | hduser@datamantra:/usr/local/confluent-3.2.0
x | hduser@datamantra:-
hduser@datamantra:~$ start-slave.sh spark://datamantra:7077
starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-datamantra.out
hduser@datamantra:~$
```

Access Spark Web UI

Check whether Spark Cluster is up and running

http://ubuntu_ip:8080

File Airflow - DAGs Fraud Alert Monitoring Dashboard Spark Master at spark://pixipanda:7077

Not secure | 192.168.0.109:8080

Apache Spark 2.2.1 Spark Master at spark://pixipanda:7077

URL: spark://pixipanda:7077
REST URL: spark://pixipanda:6066 (cluster mode)
Alive Workers: 1
Cores in use: 2 Total, 0 Used
Memory in use: 3.8 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20191023200726-192.168.0.109-36277	192.168.0.109:36277	ALIVE	2 (0 Used)	3.8 GB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

Completed Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

Start the dashboard for visualization

```
cd ~/workspace/Fraud-alert-dashboard/  
mvn exec:java -Dserver.port=8070  
-Dexec.mainClass="com.datamantra.fraudalertdashboard.dashboard.FraudAlertDa  
shboard"
```

```
hduser@datamantra:~/workspace/Fraud-alert-dashboard$ mvn exec:java -Dserver.port=8070 -Dexec.mainClass="com.datamantra.fraudalertdashboard.FraudAlertDashboard"
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building fraud-alert-dashboard 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- exec-maven-plugin:1.5.0:java (default-cli) @ fraud-alert-dashboard ---
```

Access Dashboard Web UI

http://ubuntu_ip:8070

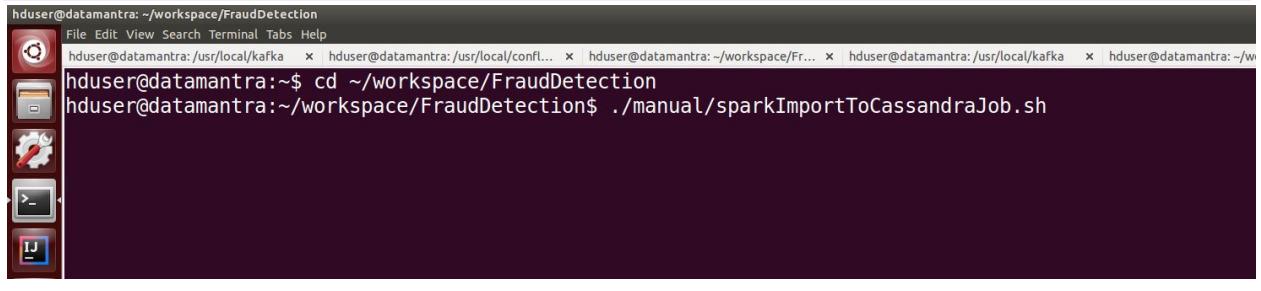
Fraud Alert Monitoring Dashboard

7.6 Airflow Automation

Start Spark Import Job

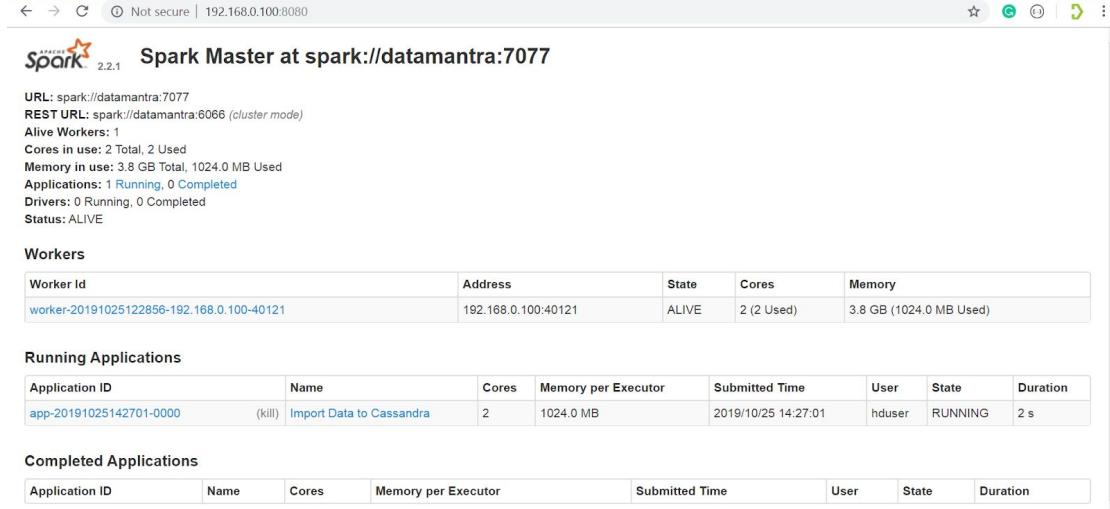
This Job will be started manually to import customer data and transaction data from the filesystem to Cassandra Keyspace.

```
cd ~/workspace/FraudDetection  
../manual/sparkImportToCassandraJob.sh
```



hduser@datamantra: ~/workspace/FraudDetection

```
File Edit View Search Terminal Tabs Help  
hduser@datamantra: /usr/local/kafka x hduser@datamantra: /usr/local/confl... x hduser@datamantra: ~/workspace/Fr... x hduser@datamantra: /usr/local/kafka x hduser@datamantra: ~/w  
hduser@datamantra:~$ cd ~/workspace/FraudDetection  
hduser@datamantra:~/workspace/FraudDetection$ ./manual/sparkImportToCassandraJob.sh
```



Spark 2.2.1

Spark Master at spark://datamantra:7077

URL: spark://datamantra:7077
REST URL: spark://datamantra:6066 (cluster mode)
Alive Workers: 1
Cores in use: 2 Total, 2 Used
Memory in use: 3.8 GB Total, 1024.0 MB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20191025122856-192.168.0.100-40121	192.168.0.100:40121	ALIVE	2 (2 Used)	3.8 GB (1024.0 MB Used)

Running Applications

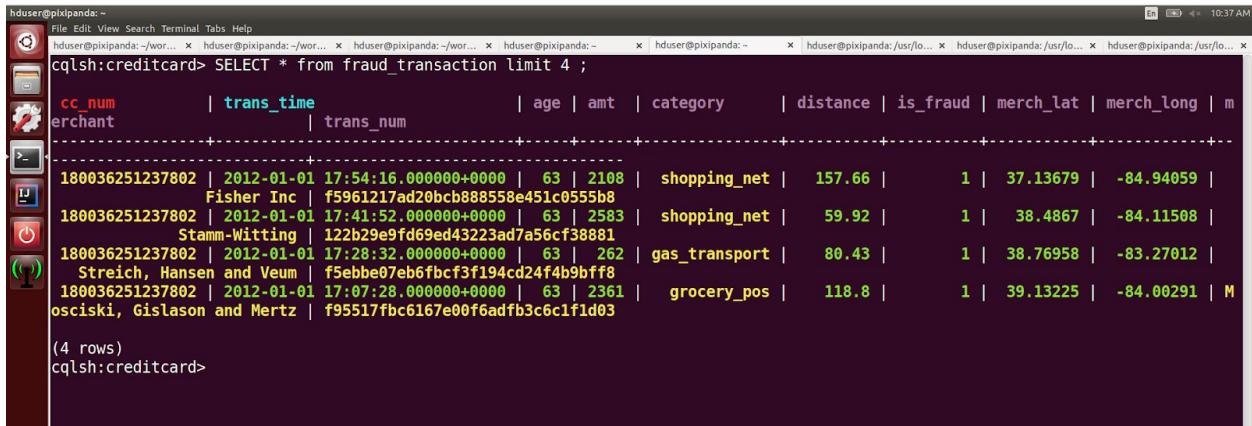
Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191025142701-0000	(kill) Import Data to Cassandra	2	1024.0 MB	2019/10/25 14:27:01	hduser	RUNNING	2 s

Completed Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

Output

Transaction data and customer data is imported to Cassandra Keyspace



cqlsh:creditcard> SELECT * from fraud_transaction limit 4 ;

cc_num	trans_time	age	amt	category	distance	is_fraud	merch_lat	merch_long	merchant	trans_num
180036251237802	2012-01-01 17:54:16.000000+0000	63	2108	shopping_net	157.66	1	37.13679	-84.94059	Fisher Inc	f5961217ad20bc888558e451c0555b8
180036251237802	2012-01-01 17:41:52.000000+0000	63	2583	shopping_net	59.92	1	38.4867	-84.11508	Stamm-Witting	122b29e9fd69ed43223ad7a56cf38881
180036251237802	2012-01-01 17:28:32.000000+0000	63	262	gas_transport	80.43	1	38.76958	-83.27012	Streich, Hansen and Veum	f5ebbe07eb6fbef3f194cd24f4b9bf8
180036251237802	2012-01-01 17:07:28.000000+0000	63	2361	grocery_pos	118.8	1	39.13225	-84.00291	Mosciski, Gislason and Mertz	f95517fbcc6167e00f6adfb3c6c1f1d03

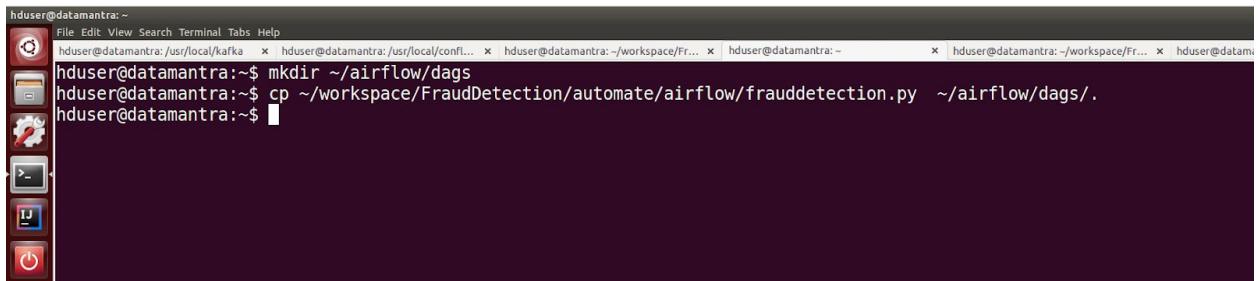
(4 rows)

cqlsh:creditcard>

Create an Airflow DAG directory.

Automation scripts must be kept in this directory. Airflow will look for scripts in this directory and it will load the scripts to the web server. So copy frauddetection.py file from FraudDetection Project to airflow dags directory

```
mkdir ~ airflow/dags
cp ~/workspace/FraudDetection/automate/airflow/frauddetection.py
~/airflow/dags/ .
```

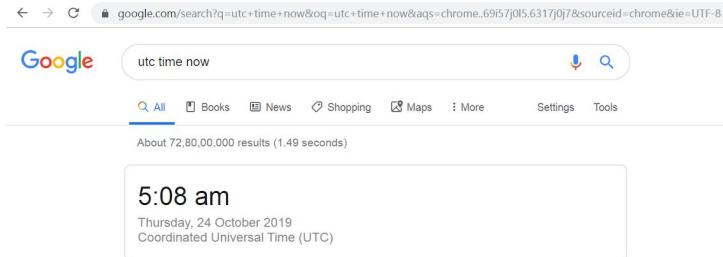


```
hduser@datamantra:~ File Edit View Search Terminal Tabs Help
hduser@datamantra:~/usr/local/kafka x hduser@datamantra:~/usr/local/conf... x hduser@datamantra:~/workspace/Fr... x hduser@datamantra:~ x hduser@datamantra:~/workspace/Fr... x hduser@datamantra:~
hduser@datamantra:~$ mkdir ~ airflow/dags
hduser@datamantra:~$ cp ~/workspace/FraudDetection/automate/airflow/frauddetection.py ~ airflow/dags/ .
hduser@datamantra:~$
```

Change the start time in [frauddetection.py](#)

Airflow uses UTC timezone

UTC time in my system is 5:08 am. We will kick start the automation at 5:20 am



start_date is set to 2019, 10, 24, 5 i.e start date is set to 5:00 am
schedule_interval is set to */20 * * * *. It means every 20 mins automation step will be evaluated.

Actual start time = start_date + schedule_interval = 5:00 + 20 mins = 5:20 am

```
default_args = {  
    'owner': 'me',  
    'start_date': dt.datetime(2019, 10, 24, 5),  
    'retries': 1,  
    'retry_delay': dt.timedelta(minutes=5),  
}  
  
with DAG(DAG_NAME,  
         default_args=default_args,  
         schedule_interval='*/20 * * * *',  
         ) as dag:
```

Compile the changes

```
python ~/airflow/dags/fraudetection.py
```

A screenshot of a terminal window titled "Terminal". The command "python ~/airflow/dags/fraudetection.py" is run, and the output shows Airflow generating grammar tables from /usr/lib/python2.7/lib2to3/Grammar.txt and /usr/lib/python2.7/lib2to3/PatternGrammar.txt. The terminal window has multiple tabs open in the background.

Reset airflow database

Reset is done so that airflow tables are created in MySQL database

```
airflow resetdb
```

Start Airflow Webserver

```
airflow webserver --port 8090
```

Access Airflow Webserver

http://ubuntu_ipaddr:8090

Click on OFF button to set it ON

The screenshot shows the Airflow web interface at the URL <http://192.168.0.109:8090/admin/>. The top navigation bar includes links for DAGs, Data Profiling, Browse, Admin, Docs, and About. The timestamp is 05:12 UTC. The main content area is titled "DAGs" and displays a table with one entry:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
		fraud_detection	*'20 ***	me				

Below the table, there is a search bar labeled "Search:" and a pagination control showing "1" of 1 entries. A link "Hide Paused DAGs" is also present.

If you click on fraud_detection you will see the automation steps.

1. Remove Model if any exist
2. Create New Model by running Spark ML Job
3. Sleep for 5 seconds (This step is optional)
4. Stop and Start Spark Streaming Job

The screenshot shows the Airflow web interface at the URL http://192.168.0.109:8090/admin/dags/fraud_detection. The top navigation bar includes links for DAGs, Data Profiling, Browse, Admin, Docs, and About. The timestamp is 05:13 UTC. The main content area shows the DAG structure:

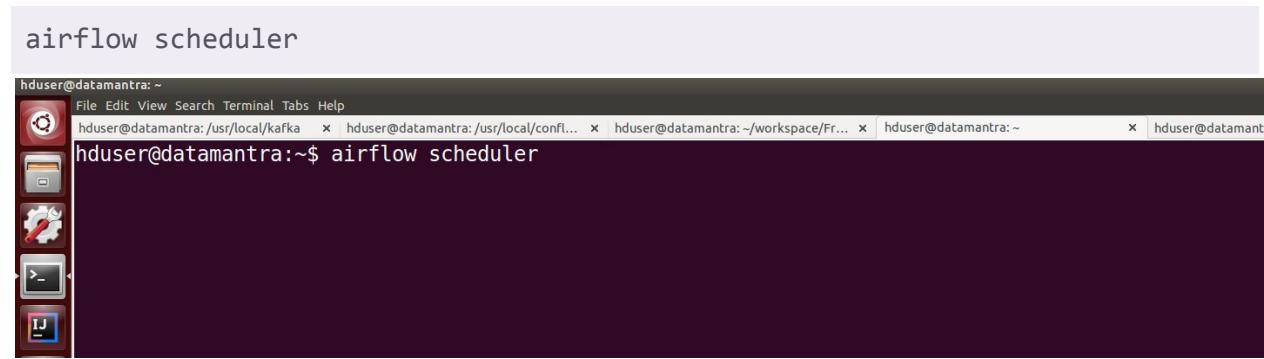
```
graph TD; DAG([DAG]) --> stop_start_streaming[stop_start_streaming]; stop_start_streaming --> sleep[sleep]; sleep --> create_model[create_model]; create_model --> remove_model[remove_model]
```

At the top, there are filters for "Base date:" and "Number of runs: 25", and a "Go" button. Below the filters, there are checkboxes for "BashOperator" and "PythonOperator". To the right, there is a legend for task status: success (green), running (yellow), failed (red), skipped (pink), retry (orange), queued (grey), and no status (white). The DAG structure is visualized as a directed graph with nodes for each task and arrows indicating dependencies.

Start Airflow Scheduler

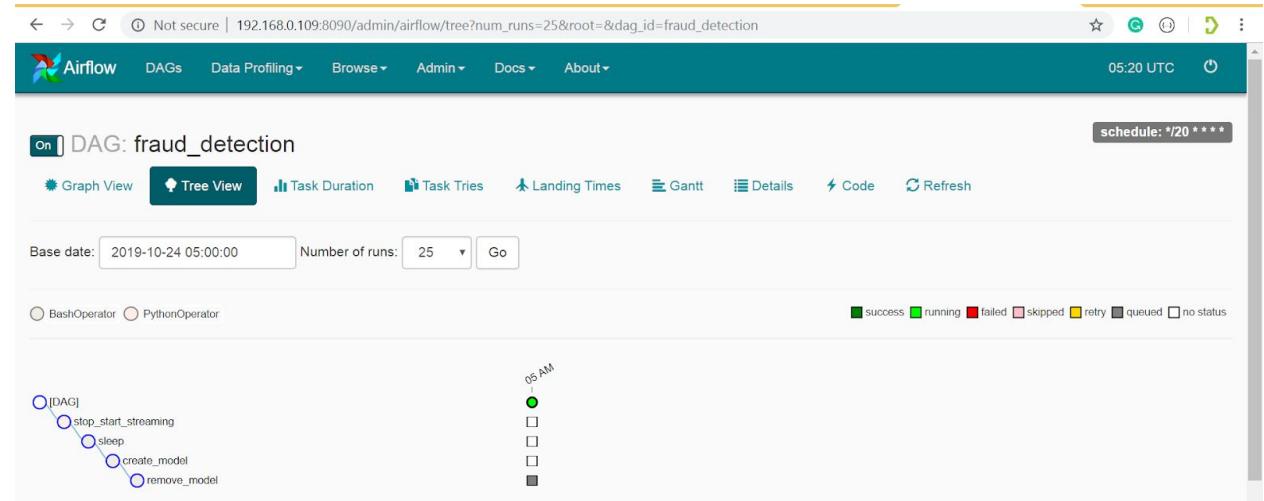
Airflow Scheduler is the one which will actually kick start your automation

```
airflow scheduler
```

A screenshot of a terminal window titled "airflow scheduler". The window shows a command-line interface with several tabs open in the background, including "File Edit View Search Terminal Tabs Help", "hduser@datamantra:/usr/local/kafka", "hduser@datamantra:/usr/local/confl...", "hduser@datamantra:~/workspace/Fr...", and "hduser@datamantra:~". The main command entered is "hduser@datamantra:~\$ airflow scheduler".

Output

Automation start at 5:20 am UTC



Spark ML Job Scheduled

← → ⌛ ⓘ Not secure | 192.168.0.109:8080 ⭐ ⓘ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂



Spark Master at spark://pixipanda:7077

URL: spark://pixipanda:7077
 REST URL: spark://pixipanda:6066 (cluster mode)
 Alive Workers: 1
 Cores in use: 3 Total, 1 Used
 Memory in use: 3.8 GB Total, 1024.0 MB Used
 Applications: 1 Running, 1 Completed
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20191024102917-192.168.0.109-46654	192.168.0.109:46654	ALIVE	3 (1 Used)	3.8 GB (1024.0 MB Used)

Running Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191024105014-0001	(kill) Fraud Detection Spark ML Training	1	1024.0 MB	2019/10/24 10:50:14	hduser	RUNNING	12 s

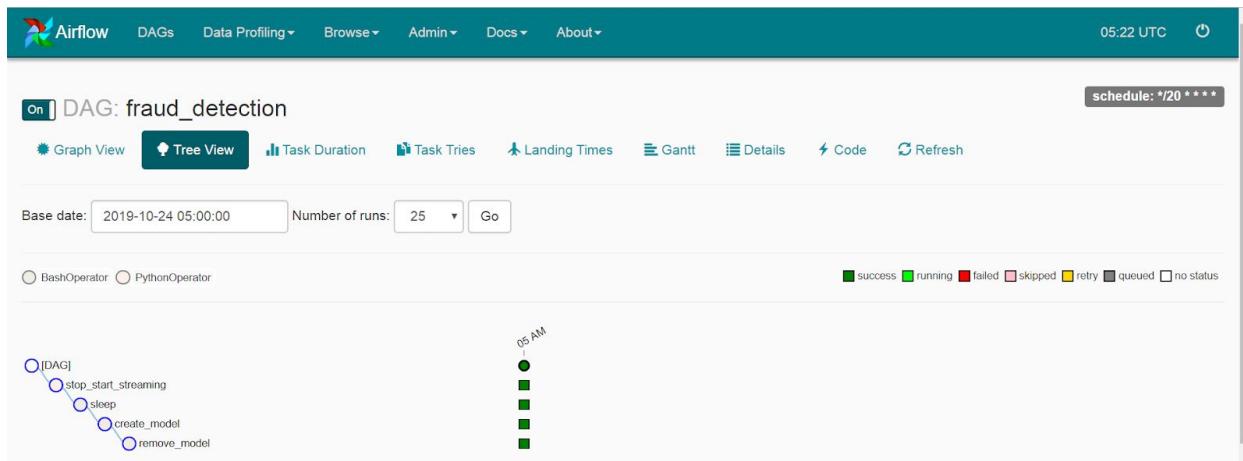
Spark ML Job Completed

Completed Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191024105014-0001	Fraud Detection Spark ML Training	1	1024.0 MB	2019/10/24 10:50:14	hduser	FINISHED	52 s
app-20191024103614-0000	Import Data to Cassandra	3	1024.0 MB	2019/10/24 10:36:14	hduser	FINISHED	30 s

All automation steps completed.

This means Spark Streaming is up and running



Spark Streaming Job is up and running

← → ⌛ ⓘ Not secure | 192.168.0.109:8080

Apache Spark 2.2.1

Spark Master at spark://pixipanda:7077

URL: spark://pixipanda:7077
 REST URL: spark://pixipanda:6066 (cluster mode)
 Alive Workers: 1
 Cores in use: 3 Total, 2 Used
 Memory in use: 3.8 GB Total, 2.0 GB Used
 Applications: 1 Running, 2 Completed
 Drivers: 1 Running, 0 Completed
 Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20191024102917-192.168.0.109-46654	192.168.0.109:46654	ALIVE	3 (2 Used)	3.8 GB (2.0 GB Used)

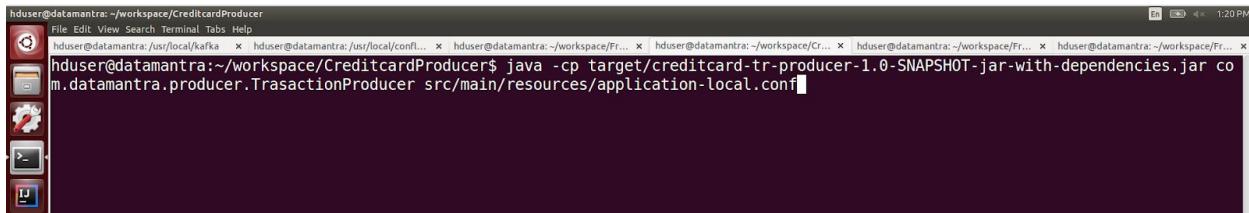
Running Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191024105132-0002	(kill) RealTime Creditcard FraudDetection	1	1024.0 MB	2019/10/24 10:51:32	hduser	RUNNING	47 s

Start Kafka Producer

Spark Streaming Job is up and running, now we will start Kafka Producer

```
java -cp
target/creditcard-tr-producer-1.0-SNAPSHOT-jar-with-dependencies.jar
com.datamantra.producer.TrasactionProducer
src/main/resources/application-local.conf
```



```
hduser@datamantra:~/workspace/CreditcardProducer$ java -cp target/creditcard-tr-producer-1.0-SNAPSHOT-jar-with-dependencies.jar com.datamantra.producer.TrasactionProducer src/main/resources/application-local.conf
```

Output

A fraud alert is displayed on the dashboard

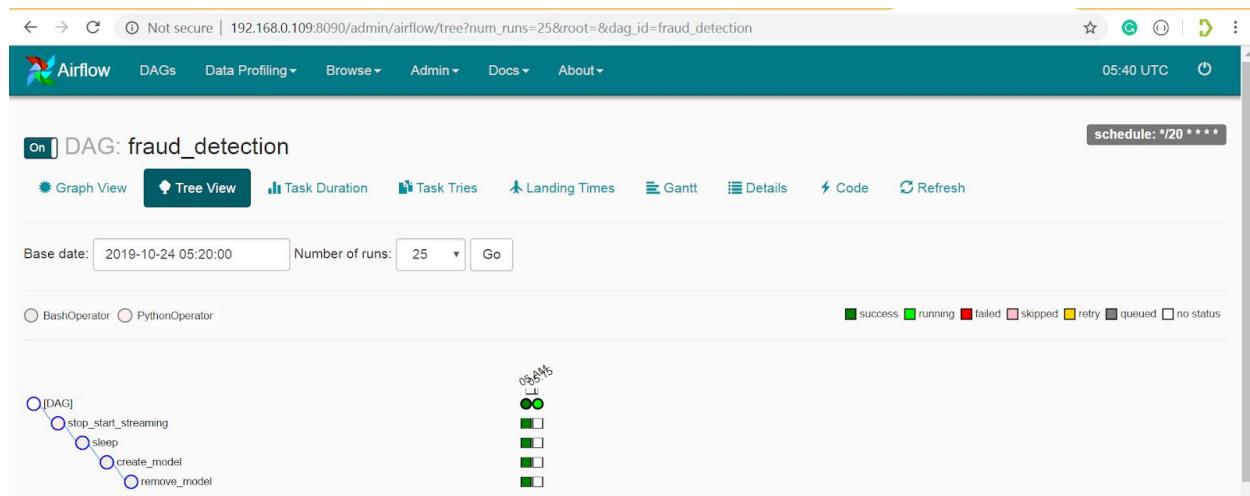
My Drive - Google Drive | Creditcard Fraud detection | Creditcard Fraud Detection | Fraud Alert Monitoring | Spark Master at spark://pixipanda:7077 | Airflow - DAGs | + | - | × | ⌛ ⓘ Not secure | 192.168.0.109:8080

Fraud Alert Monitoring Dashboard

cc_num	trans_time	trans_num	category	merchant	amt	distance	age
374115112731710	2019-10-24 10:54:17	11efb54624f40fe0...	entertainment	Johns-Hoeger	64	4.16	41

Next set of automation

Next set of automation steps scheduled at 5:40 am(every 20 mins)



Spark ML Job is running

This will create new model

Apache Spark 2.2.1

Spark Master at spark://pixipanda:7077

URL: spark://pixipanda:7077
REST URL: spark://pixipanda:8066 (cluster mode)
Alive Workers: 1
Cores in use: 3 Total, 3 Used
Memory in use: 3.8 GB Total, 3.0 GB Used
Applications: 2 Running, 2 Completed
Drivers: 1 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20191024102917-192.168.0.109-46654	192.168.0.109:46654	ALIVE	3 (3 Used)	3.8 GB (3.0 GB Used)

Running Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191024111015-0003 (kill)	Fraud Detection Spark ML Training	1	1024.0 MB	2019/10/24 11:10:15	hduser	RUNNING	6 s
app-20191024105132-0002 (kill)	RealTime Creditcard FraudDetection	1	1024.0 MB	2019/10/24 10:51:32	hduser	RUNNING	19 min

Completed Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191024111015-0003	Fraud Detection Spark ML Training	1	1024.0 MB	2019/10/24 11:10:15	hduser	FINISHED	1.1 min
app-20191024105014-0001	Fraud Detection Spark ML Training	1	1024.0 MB	2019/10/24 10:50:14	hduser	FINISHED	52 s
app-20191024103614-0000	Import Data to Cassandra	3	1024.0 MB	2019/10/24 10:36:14	hduser	FINISHED	30 s

Spark Streaming Job Stopped

Completed Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191024105132-0002	RealTime Creditcard FraudDetection	1	1024.0 MB	2019/10/24 10:51:32	hduser	FINISHED	20 min
app-20191024111015-0003	Fraud Detection Spark ML Training	1	1024.0 MB	2019/10/24 11:10:15	hduser	FINISHED	1.1 min
app-20191024105014-0001	Fraud Detection Spark ML Training	1	1024.0 MB	2019/10/24 10:50:14	hduser	FINISHED	52 s
app-20191024103614-0000	Import Data to Cassandra	3	1024.0 MB	2019/10/24 10:36:14	hduser	FINISHED	30 s

New Streaming Job Started

This will load the new model created by the previous Spark ML Job. Transactions will be predicted using this new model.

Screenshot of the Spark Master UI at `spark://pixipanda:7077`. The UI shows the following information:

- URL:** `spark://pixipanda:7077`
- REST URL:** `spark://pixipanda:6066 (cluster mode)`
- Alive Workers:** 1
- Cores in use:** 3 Total, 2 Used
- Memory in use:** 3.8 GB Total, 2.0 GB Used
- Applications:** 1 Running, 4 Completed
- Drivers:** 1 Running, 1 Completed
- Status:** ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20191024102917-192.168.0.109-46654	192.168.0.109:46654	ALIVE	3 (2 Used)	3.8 GB (2.0 GB Used)

Running Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191024111156-0004 (kill)	RealTime Creditcard FraudDetection	1	1024.0 MB	2019/10/24 11:11:56	hduser	RUNNING	25 s

Screenshot of the Airflow UI showing the DAG `fraud_detection`. The DAG has the following tasks:

- [DAG]
- `stop_start_streaming`
- `sleep`
- `create_model`
- `remove_model`

The DAG is scheduled to run every 20 minutes. The current run status is shown in the Gantt chart:

Task	Run 1	Run 2	Run 3	Run 4	Run 5
[DAG]	success	success	success	success	success
stop_start_streaming	running	running	running	running	running
sleep	success	success	success	success	success
create_model	success	success	success	success	success
remove_model	success	success	success	success	success