Big Data Technology - Final Project

The revolution of electric vehicle

Professor:

Mrudula Mukadam

Team:

Ba Luan Tran

Emelyne Kwizera

Trong Hoang

I. Dataset:

1. Download dataset

- Download latest dataset from link: https://data.wa.gov/Demographics/Electric-Vehicles-by-Manufacturer/958t-wifj
- Put the dataset file into project's folder "input" to simulate streaming from data source.

2. Dataset information

- File format: CSV
- Dataset size: 112634 rows
- Description: This dataset shows the Battery Electric Vehicles (BEVs) and Plug-in Hybrid Electric Vehicles (PHEVs) that are currently registered through Washington State Department of Licensing (DOL).

II. Kafka:

1. Install and start Kafka server:

a. Download Kafka and extract:

- Download link:
  https://kafka.apache.org/downloads
- Project version:
  https://archive.apache.org/dist/kafka/2.4.1/kafka_2.12-2.4.1.tgz
- Extract the tgz file and change directory to
  $ cd /home/cloudera/Downloads/kafka_2.12-2.4.1

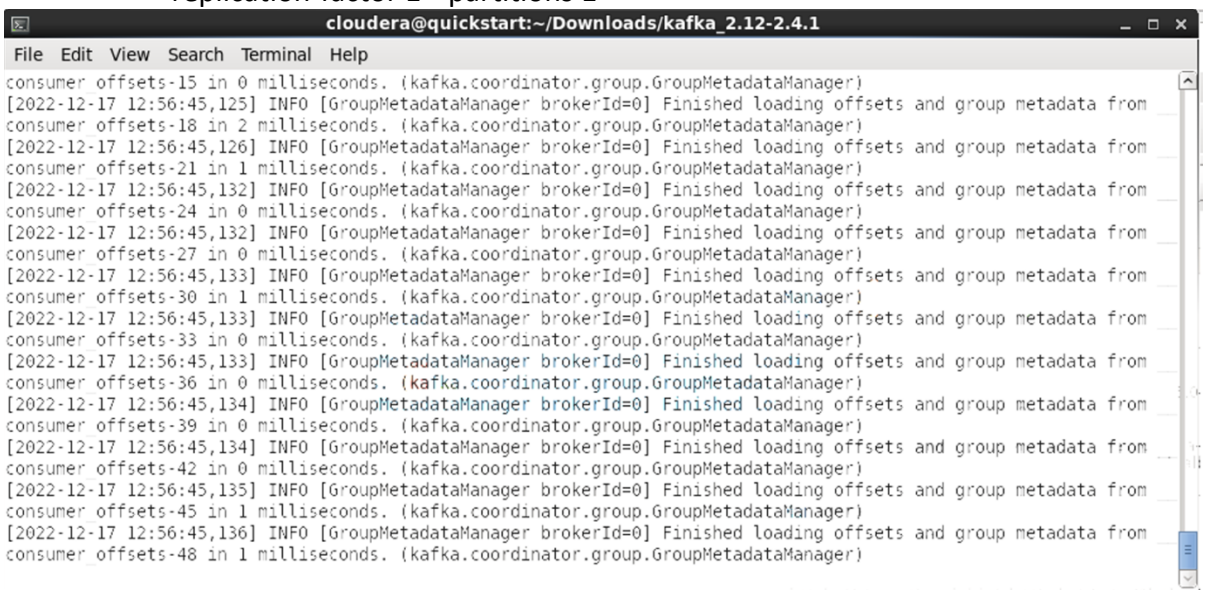
b. Start Zookeeper and Kafka server

- Currently, the Zookeeper server already started in the cloudera-quickstart. If it's not start, we can use the command to start Zookeeper:

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

- Start Kafka server:

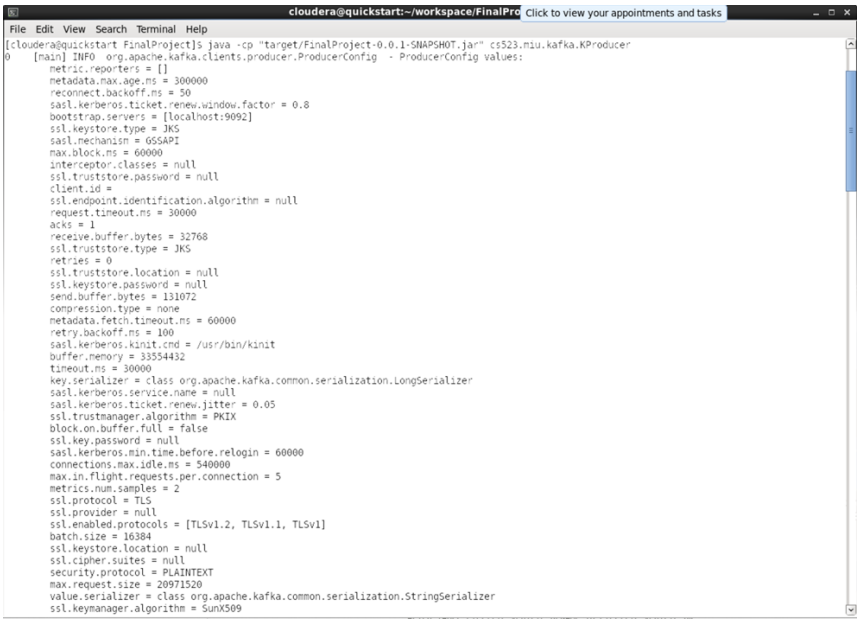
  ./bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic vehicle-topic --replication-factor 1 --partitions 1



2. **Run Kafka Producer:**
    a. **Publish directly a file**

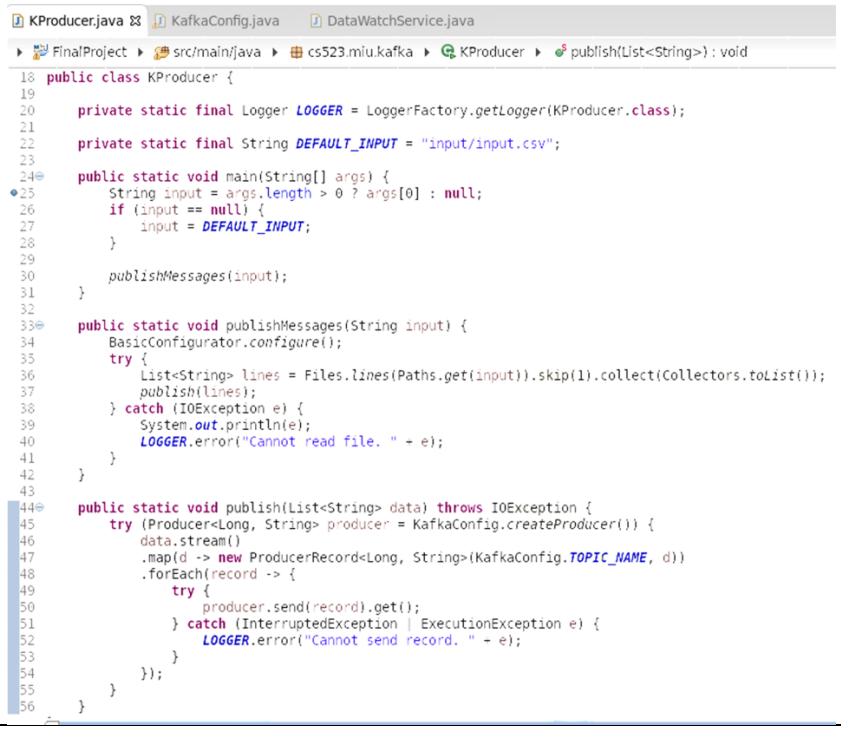
- Run the command:

  java -cp "target/FinalProject-0.0.1-SNAPSHOT.jar" cs523.miu.kafka.KProducer



- After the Kafka Producer started, it will read the input file in the path /input and cast to string list in Java and send each line message to the topic **"vehicle-topic"**

- Java code:



b. Use WatchService to listen the file in a directory
(Refer to: https://howtodoinjava.com/java8/java-8-watchservice-api-tutorial/?utm_campaign=javaswag&utm_medium=web&utm_source=javaswag15**)**

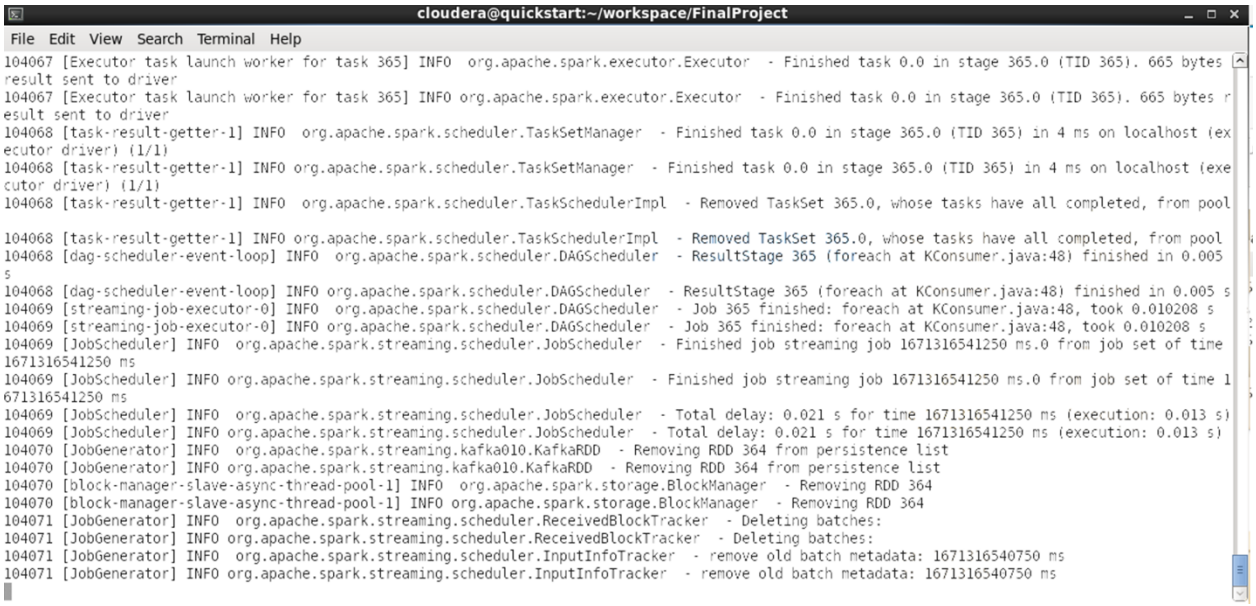
- Run the command:
  java -cp "target/FinalProject-0.0.1-SNAPSHOT.jar" cs523.miu.utils.DataWatchService

- If we add the input file to the /input directory, the WatchService will be read the file and cast to string list in Java and send each line message to the topic **"vehicle-topic"**

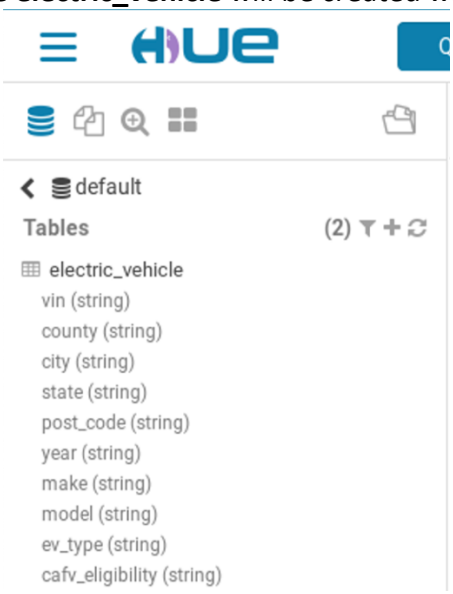
3. Run Kafka Consumer:

- Run the command

java -cp "target/FinalProject-0.0.1-SNAPSHOT.jar" cs523.miu.App



- After the Kafka Consumer started, the **electric_vehicle** will be created with empty data



- When the Kafka Consumer received messages from **the topic "vehicle-topic"** and the consumer will be read and inserted the data to the Hive table

	electric_vehicle.vin	electric_vehicle.county	electric_vehicle.city	electric_vehicle.state	electric_vehicle.post_code	electric_vehic
1	JTMEB3FV6N	Monroe	Key West	FL	33040	2022
2	1G1RD6E45D	Clark	Laughlin	NV	89029	2013
3	JN1AZ0CP8B	Yakima	Yakima	WA	98901	2011
4	1G1FW6S08H	Skagit	Concrete	WA	98237	2017
5	3FA6P0SU1K	Snohomish	Everett	WA	98201	2019
6	5YJ3E1EB5J	Snohomish	Bothell	WA	98021	2018
7	1N4AZ0CP4D	Snohomish	Everett	WA	98203	2013
8	1N4AZ0CP0D	Snohomish	Mukilteo	WA	98275	2013
9	1N4BZ0CP4G	Island	Clinton	WA	98236	2016
10	KNDJP3AE2G	Skagit	Anacortes	WA	98221	2016

- Java code:

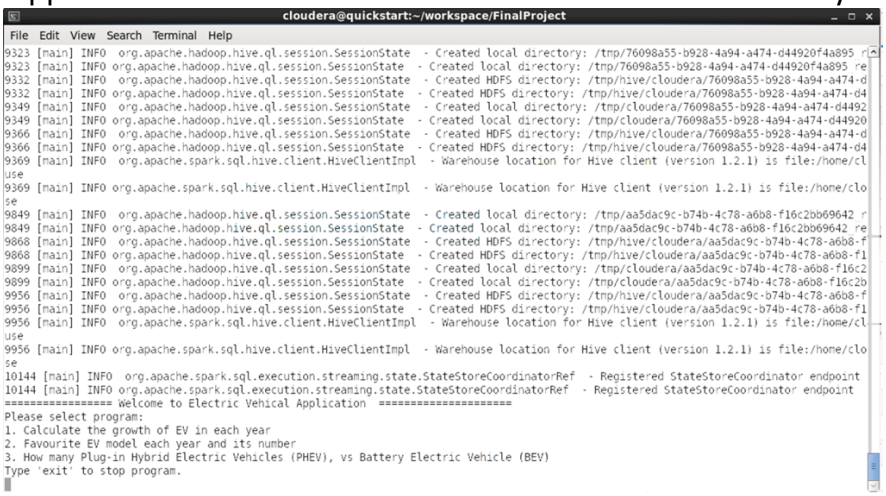
```
public static void startConsumer() throws InterruptedException {
    try {
        HiveRepository repo = HiveRepository.getInstance();

        JavaStreamingContext streamingContext = new JavaStreamingContext(
                new SparkConf().setAppName("SparkStreaming").setMaster("local[*]"), new Duration(250));

        Map<String, Object> kafkaParams = new HashMap<>();
        kafkaParams.put("bootstrap.servers", "localhost:9092");
        kafkaParams.put("key.deserializer", StringDeserializer.class);
        kafkaParams.put("value.deserializer", StringDeserializer.class);
        kafkaParams.put("group.id", "group");
        kafkaParams.put("auto.offset.reset", "latest");
        kafkaParams.put("enable.auto.commit", false);

        Collection<String> topics = Arrays.asList(KafkaConfig.TOPIC_NAME);

        JavaInputDStream<ConsumerRecord<String, String>> stream =
          KafkaUtils.createDirectStream(
            streamingContext,
            LocationStrategies.PreferConsistent(),
            ConsumerStrategies.<String, String>Subscribe(topics, kafkaParams)
          );

        // Insert data into Hive from Kafka
        stream.foreachRDD(rdd -> rdd.foreach(x -> {
            VehicleRecord record = RecordParser.parse(x.value());
            if (record != null) {
                System.out.println(x.value());
                System.out.println(record);
                repo.insertRecord(record);
            }
        }));
        streamingContext.start();
        streamingContext.awaitTermination();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

III. Record Analysis:

1. Start Terminal and run the command:

java -cp "target/FinalProject-0.0.1-SNAPSHOT.jar" cs523.miu.sparksql.RecordAnalysis

- The RecordAnalysis application will be executed and we can choose some analysis functions



- Java code:

```
public class RecordAnalysis {
    private static final Logger LOGGER = LoggerFactory.getLogger(RecordAnalysis.class);

    private static SparkSession session;

    public static void init() throws IOException, AnalysisException {
        final SparkConf sparkConf = new SparkConf();

        sparkConf.setMaster("local");
        sparkConf.set("hive.metastore.uris", "thrift://localhost:9083");
        session = SparkSession
                .builder()
                .appName("Spark SQL-Hive")
                .config(sparkConf)
                .enableHiveSupport()
                .getOrCreate();
    }
```

```
private static void printMenu() {
    System.out.println("================== Welcome to Electric Vehicle Application ====================");
    System.out.println("Please select program:");
    System.out.println("1. Calculate the growth of EV in each year ");
    System.out.println("2. Favourite EV model each year and its number");
    System.out.println("3. How many Plug-in Hybrid Electric Vehicles (PHEV), vs Battery Electric Vehicle (BEV)");
    System.out.println("Type 'exit' to stop program.");
}


public static void main(String[] args) throws IOException, AnalysisException {
    BasicConfigurator.configure();

    try (Scanner scanner = new Scanner(System.in)) {
        init();

        while (true) {
            printMenu();
            String option = scanner.nextLine();

            switch (option) {
                case "1":
                    getNumberOfVehiclesEachYear();
                    break;
                case "2":
                    getEVModelEachYear();
                    break;
                case "3":
                    getVehicleTypeGroup();
                    break;
                case "exit":
                    System.exit(1);
                default:
            }
        }

    } catch (IOException e) {
        LOGGER.info("================== PROCESSING TO EXIT APPLICATION ... ====================");
        LOGGER.error("An error occur while running Electric Vehicle Application. " + e);
```

a. Program 1:

- The Program 1 will analyze data from electric_vehicle table and create **vehicle_count_analysis** table and insert annalysis data to the table

	vehicle_count_analysis.year	vehicle_count_analysis.count
1	2022	2
2	2021	2
3	2020	8
4	2019	9
5	2018	11
6	2017	6
7	2016	11
8	2015	2
9	2014	1
10	2013	9
11	2011	1

- Java code

```
public static List<VehiclesEachYearRecord> getNumberOfVehiclesEachYear() {
    StringBuilder sql = new StringBuilder();
    sql.append("SELECT year, COUNT(1) AS count ")
        .append(" FROM ").append(HiveRepository.TABLE_NAME)
        .append(" GROUP BY year ")
        .append(" ORDER BY year DESC ");

    session.sql("DROP TABLE IF EXISTS vehicle_count_analysis ");
    Dataset<Row> sqlDF = session.sql(sql.toString());
    sqlDF.write().saveAsTable("vehicle_count_analysis");
    sqlDF.show();

    List<VehiclesEachYearRecord> list = sqlDF.as(Encoders.bean(VehiclesEachYearRecord.class)).collectAsList();
    return list;
}
```

b. Program 2:

- The Program 1 will analyze data from electric_vehicle table and create **vehicle_model_analysis** table and insert annalysis data to the table

	vehicle_model_analysis.make	vehicle_model_analysis.model	vehicle_model_analysis.evtype	vehicle_model_analysis.count
1	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	1
2	NISSAN	LEAF	Battery Electric Vehicle (BEV)	15
3	CHEVROLET	VOLT	Plug-in Hybrid Electric Vehicle (PHEV)	3
4	AUDI	A3	Plug-in Hybrid Electric Vehicle (PHEV)	1
5	HONDA	CLARITY	Plug-in Hybrid Electric Vehicle (PHEV)	1
6	AUDI	Q5 E	Plug-in Hybrid Electric Vehicle (PHEV)	1
7	TESLA	MODEL S	Battery Electric Vehicle (BEV)	3
8	BMW	I3	Plug-in Hybrid Electric Vehicle (PHEV)	2
9	KIA	SOUL EV	Battery Electric Vehicle (BEV)	1
10	FORD	FUSION	Plug-in Hybrid Electric Vehicle (PHEV)	1
11	FORD	F-150	Battery Electric Vehicle (BEV)	1
12	TESLA	MODEL X	Battery Electric Vehicle (BEV)	3

- Java code:

```
public static List<VehicleTypeReport> getVehicleTypeGroup() {
    StringBuilder sql = new StringBuilder();
    sql.append("SELECT ev_type AS evType, COUNT(1) AS count ")
        .append(" FROM ").append(HiveRepository.TABLE_NAME)
        .append(" GROUP BY ev_type");

    session.sql("DROP TABLE IF EXISTS vehicle_type_analysis ");
    Dataset<Row> sqlDF = session.sql(sql.toString());
    sqlDF.write().saveAsTable("vehicle_type_analysis");
    sqlDF.show();

    List<VehicleTypeReport> list = sqlDF.as(Encoders.bean(VehicleTypeReport.class)).collectAsList();
    return list;
}
```

c. Program 3:

- The Program 1 will analyze data from electric_vehicle table and create **vehicle_type_analysis** table and insert annalysis data to the table

	vehicle_type_analysis.evtype	vehicle_type_analysis.count
1	Plug-in Hybrid Electric Vehicle (PHEV)	17
2	Battery Electric Vehicle (BEV)	45

- Java code:

```
public static List<FavouriteEVModelRecord> getEVModelEachYear() {
    StringBuilder sql = new StringBuilder();
    sql.append("SELECT make, model, ev_type AS evType, COUNT(1) AS count ")
        .append(" FROM ")
        .append(HiveRepository.TABLE_NAME)
        .append(" GROUP BY make, model, ev_type ");

    session.sql("DROP TABLE IF EXISTS vehicle_model_analysis ");
    Dataset<Row> sqlDF = session.sql(sql.toString());
    sqlDF.write().saveAsTable("vehicle_model_analysis");
    sqlDF.show();

    List<FavouriteEVModelRecord> list = sqlDF.as(Encoders.bean(FavouriteEVModelRecord.class)).collectAsList();
    return list;
}
```

2. HIVE AND SPARK SQL
    a. Create configurations to enable spark to connect to hive

```
SparkSession spark = SparkSession

.builder()

.master("local[*]")

.appName("Java Spark Hive Example")

.config("hive.metastore.warehouse.dir", "/user/hive/warehouse")

.config("hive.metastore.uris", "thrift://localhost:9083")

.config("spark.sql.warehouse.dir", "/user/hive/warehouse")

.config("hive.exec.scratchdir",

"/tmp/a-folder-that-the-current-user-has-permission-to-write-in")

.config("spark.yarn.security.credentials.hive.enabled", "true")

.config("spark.sql.hive.metastore.jars", "maven")

.config("spark.sql.hive.metastore.version", "1.2.1")

.config("spark.sql.catalogImplementation", "hive")

.enableHiveSupport()

.getOrCreate();
```

    b. Create a pom.xml file and add hive and sparksql dependency and run mvn clean install.

*JavaSparkHiveExample.java *SparkSQL_Hive/pom.xml

```
  <version>0.0.1-SNAPSHOT</version>

   <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <dependencies>
  <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql -->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.12</artifactId>
    <version>3.0.1</version>
    <scope>provided</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql-kafka-0-10 -->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql-kafka-0-10_2.13</artifactId>
    <version>3.3.1</version>
    <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-hive -->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-hive_2.12</artifactId>
    <version>3.0.1</version>
    <scope>compile</scope>
</dependency>
  </dependencies>
```

c. Create tables and Query in hive table to find insight

```
spark.sql("CREATE EXTERNAL TABLE electricVehicles(vin String, county String,city String, "
        + "state String, post_code int, year int, make String, "
        + "model String, ev_type String,cafv_eligibility String) "
        + "LOCATION 'hdfs://localhost:8020/user/cloudera/input'"
        + " ROW FORMAT DELIMITED " + "FIELDS TERMINATED BY ','");

//queries

Dataset<Row> carYear = spark.sql("select year, COUNT(*) as count from electricVehicles group by year");
carYear.write().saveAsTable("countcar");

Dataset<Row> vehicleType = spark.sql("select model, ev_type  from electricVehicles");
vehicleType.write().saveAsTable("vehicleType");

//chaining of dataset

Dataset<Row> modelYear = spark.sql("select model, year, COUNT(*) as count from electricVehicles group by model,year");
modelYear.createOrReplaceTempView("modelYear");
Dataset<Row> table = spark.sql("select *from modelYear where count>200 ");
table.show();

spark.sql("SELECT * FROM electricVehicles LIMIT 10").show();
//close spark connection
spark.close();
```

d. Chaining of Datasets

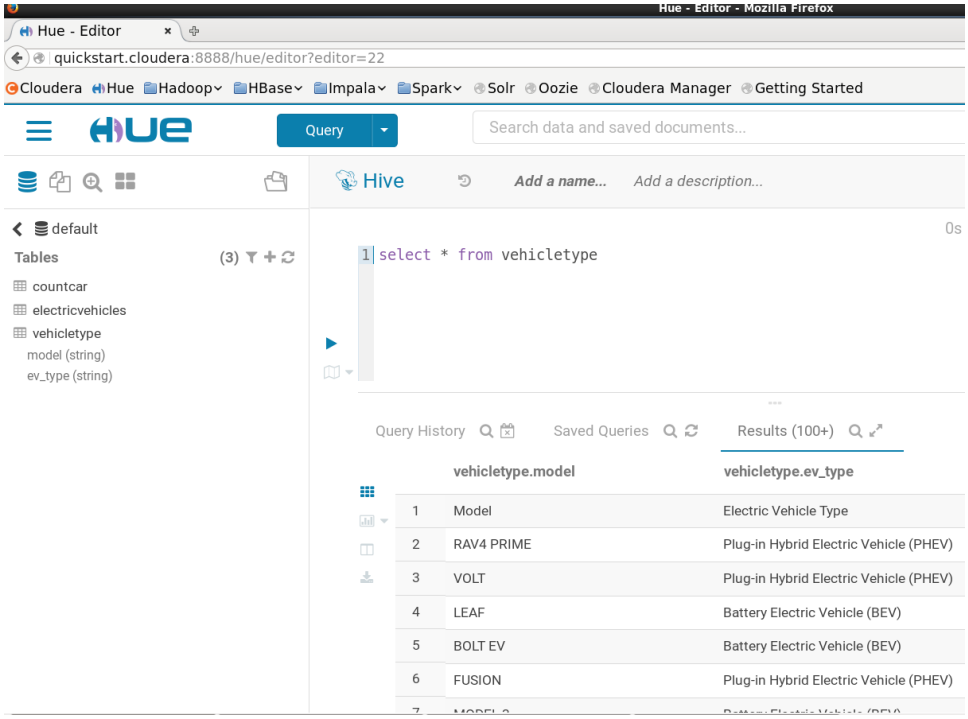
```
64
65        //chaining of dataset
66
67        Dataset<Row> modelYear = spark.sql("select model, year, COUNT(*) as count from electricVehicles group by model,year");
68        modelYear.createOrReplaceTempView("modelYear");
69        Dataset<Row> table = spark.sql("select *from modelYear where count>200 ");
70        table.show();
71
72        spark.sql("SELECT * FROM electricVehicles LIMIT 10").show();
73        //close spark connection
74        spark.close();
```

Problems Javadoc Declaration Console

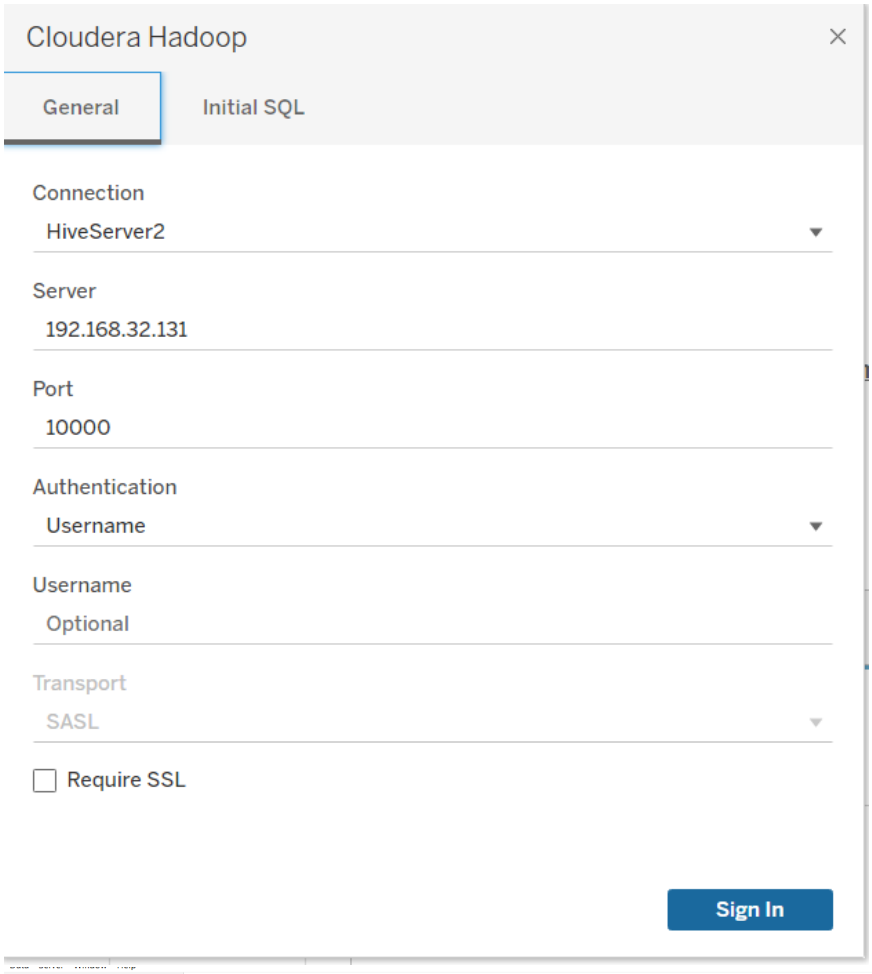
<terminated> SparkSqlHiveExample [Java Application] /usr/java/jdk1.8.0_181/bin/java (Dec 19, 2022, 9:11:55 AM)

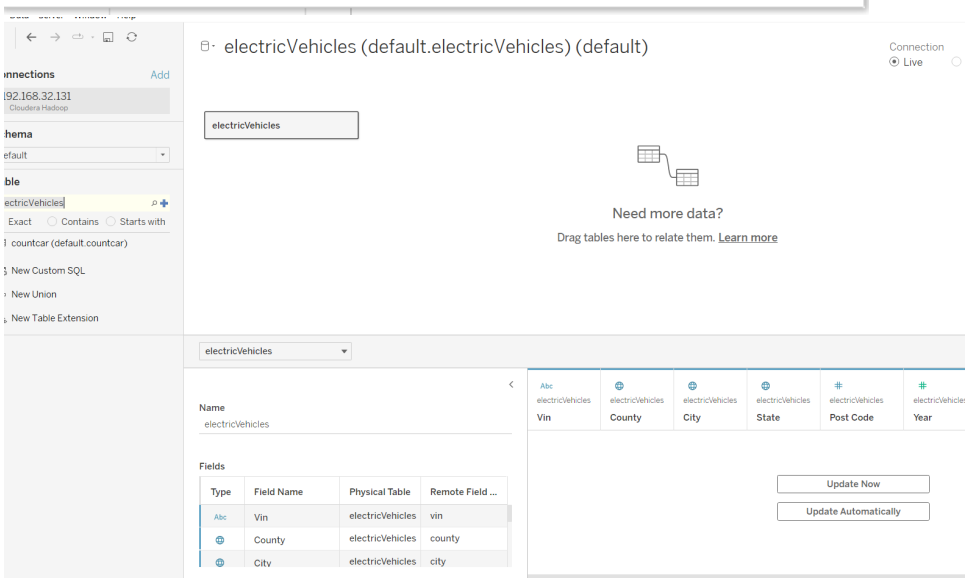
```
|      model|year|count|
+-----------+----+-----+
|     BOLT EV|2019|  856|
|     FUSION|2017|  358|
|       LEAF|2021|  534|
|         X3|2021|  212|
|       NIRO|2020|  522|
|       VOLT|2015|  369|
|       ID.4|2021|  988|
|      C-MAX|2013|  392|
|       NIRO|2022|  751|
|PRIUS PRIME|2020|  311|
|       LEAF|2011|  755|
|    BOLT EV|2022|  892|
|    BOLT EV|2018|  423|
|    BOLT EV|2021|  377|
|    BOLT EV|2017| 1297|
|    MODEL Y|2022| 7731|
|       XC40|2021|  245|
|    SORENTO|2022|  345|
```

e. After running the queries, the tables are created



f. Connect tableau to the database





g. Visualize trends and analysis