

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-wifi>
- 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
```



A terminal window titled "cloudera@quickstart:~/Downloads/kafka_2.12-2.4.1" displaying the output of a Kafka topic creation command. The log shows multiple INFO messages from the GroupMetadataManager indicating the finished loading of offsets and group metadata for various consumer offsets (from -15 to -48) in 0 to 1 milliseconds.

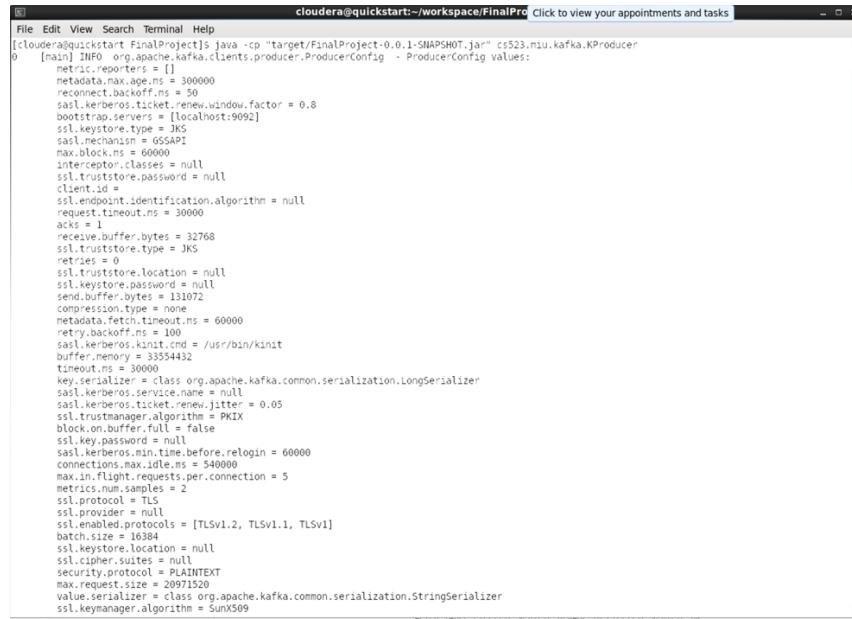
```
consumer offsets-15 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,125] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-18 in 2 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,126] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-21 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,132] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-24 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,132] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-27 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,133] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-30 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,133] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-33 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,133] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-36 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,134] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-39 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,134] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-42 in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,135] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-45 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2022-12-17 12:56:45,136] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from consumer offsets-48 in 1 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
```

2. Run Kafka Producer:

a. Publish directly a file

- Run the command:

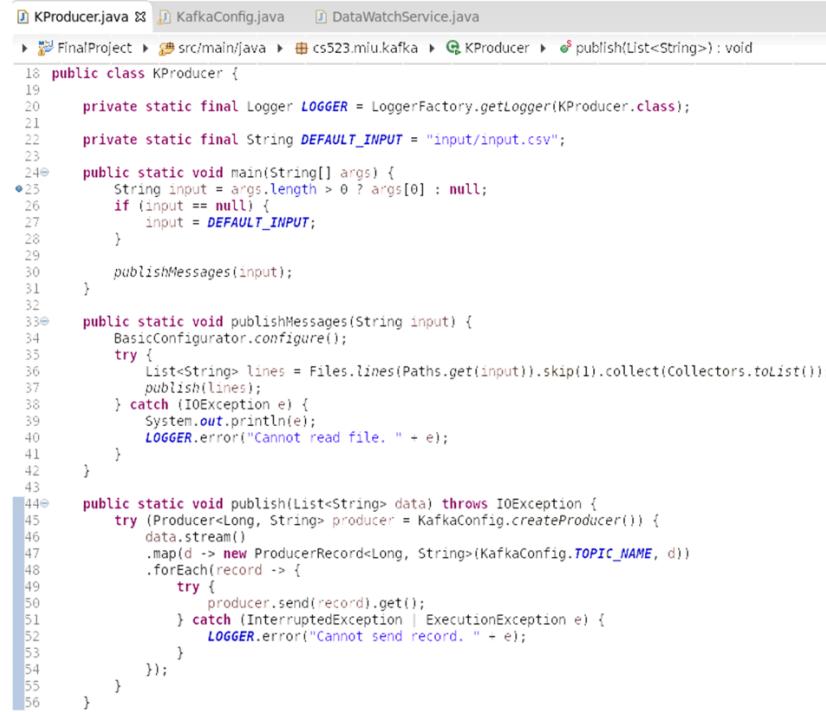
```
java -cp "target/FinalProject-0.0.1-SNAPSHOT.jar" cs523.miu.kafka.KProducer
```



```
[cloudera@quickstart:~/workspace/FinalProj] java -cp "target/FinalProject-0.0.1-SNAPSHOT.jar" cs523.miu.kafka.KProducer
[main] INFO org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
    netty.config = null
    netty.max.apgs.ms = 300000
    reconnect.backoff.ms = 50
    sasl.kerberos.ticket.renew.window.factor = 0.8
    bootstrap.servers = [localhost:9092]
    ssl.keystore.type = JKS
    sasl.mechanism = GSSAPI
    max.block.ms = 60000
    interceptor.classes = null
    ssl.truststore.password = null
    client.id = null
    sasl.username.identification.algorithm = null
    request.timeout.ms = 30000
    acks = 1
    receive.buffer.bytes = 32768
    ssl.truststore.type = JKS
    retries = 0
    ssl.truststore.location = null
    ssl.truststore.password = null
    send.buffer.bytes = 131072
    compression.type = none
    metadata.fetch.timeout.ms = 60000
    retry.backoff.ms = 100
    sasl.kerberos.kinit.cmd = /usr/bin/kinit
    buffer.memory = 33554432
    timeout.ms = 30000
    key.serializer = class org.apache.kafka.common.serialization.LongSerializer
    sasl.kerberos.service.name = null
    sasl.kerberos.ticket.renew.jitter = 0.05
    sasl.trustmanager.algorithm = PKIX
    block.on.buffer.full = false
    sasl.password = null
    sasl.kerberos.min.time.before.relogin = 60000
    connections.max.idle.ms = 540000
    max.in.flight.requests.per.connection = 5
    metrics.num.samples = 2
    ssl.protocol = TLS
    ssl.enabled.protocols = [TLSv1.2, TLSv1.1, TLSv1]
    batch.size = 16384
    ssl.keystore.location = null
    ssl.cipher.suites = null
    security.protocol = PLAINTEXT
    max.request.size = 20971520
    value.serializer = class org.apache.kafka.common.serialization.StringSerializer
    sasl.keymanager.algorithm = SunX509
```

- 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:



```
KProducer.java ✘ KafkaConfig.java ✘ DataWatchService.java
▶ FinalProject > src/main/java > cs523.miu.kafka > KProducer > publish(List<String>) : void

18 public class KProducer {
19
20     private static final Logger LOGGER = LoggerFactory.getLogger(KProducer.class);
21
22     private static final String DEFAULT_INPUT = "input/input.csv";
23
24     public static void main(String[] args) {
25         String input = args.length > 0 ? args[0] : null;
26         if (input == null) {
27             input = DEFAULT_INPUT;
28         }
29
30         publishMessages(input);
31     }
32
33     public static void publishMessages(String input) {
34         BasicConfigurator.configure();
35         try {
36             List<String> lines = Files.lines(Paths.get(input)).skip(1).collect(Collectors.toList());
37             publish(lines);
38         } catch (IOException e) {
39             System.out.println(e);
40             LOGGER.error("Cannot read file. " + e);
41         }
42     }
43
44     public static void publish(List<String> data) throws IOException {
45         try (Producer<Long, String> producer = KafkaConfig.createProducer()) {
46             data.stream()
47                 .map(d -> new ProducerRecord<Long, String>(KafkaConfig.TOPIC_NAME, d))
48                 .forEach(record -> {
49                     try {
50                         producer.send(record).get();
51                     } catch (InterruptedException | ExecutionException e) {
52                         LOGGER.error("Cannot send record. " + e);
53                     }
54                 });
55         }
56     }
}
```

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**”



The screenshot shows a terminal window titled "cloudera@quickstart:~/workspace/FinalProject". The window displays a large amount of Kafka configuration code and its execution log.

```
acks = 1
receive.buffer.bytes = 32768
ssl.truststore.type = JKS
retries = 0
ssl.truststore.location = null
ssl.keystore.password = null
send.buffer.bytes = 131072
compression.type = none
metadata.fetch.timeout.ms = 60000
retry.backoff.ms = 100
sasl.kerberos.kinit.cmd = /usr/bin/kinit
buffer.memory = 33554432
timeout.ms = 30000
key.serializer = class org.apache.kafka.common.serialization.LongSerializer
sasl.kerberos.service.name = null
sasl.kerberos.ticket.renew.jitter = 0.05
ssl.trustmanager.algorithm = PKIX
block.on.buffer.full = false
ssl.key.password = null
sasl.kerberos.min.time.before.relogin = 60000
connections.max.idle.ms = 540000
max.in.flight.requests.per.connection = 5
metrics.num.samples = 2
ssl.protocol = TLS
ssl.provider = null
ssl.enabled.protocols = [TLSv1.2, TLSv1.1, TLSv1]
batch.size = 16384
ssl.keystore.location = null
ssl.cipher.suites = null
security.protocol = PLAINTEXT
max.request.size = 20971520
value.serializer = class org.apache.kafka.common.serialization.StringSerializer
ssl.keymanager.algorithm = SunX509
metrics.sample.window.ms = 30000
partitioner.class = class org.apache.kafka.clients.producer.internals.DefaultPartitioner
linger.ms = 0

879 [main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka version : 0.10.0.1
879 [main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka version : 0.10.0.1
879 [main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka version : 0.10.0.1
879 [main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka commitId : a7a17cdec9eaa6c5
879 [main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka commitId : a7a17cdec9eaa6c5
879 [main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka commitId : a7a17cdec9eaa6c5
1001 [main] INFO org.apache.kafka.clients.producer.KafkaProducer - Closing the Kafka producer with timeoutMillis = 9223372036854775807 ms.
1001 [main] INFO org.apache.kafka.clients.producer.KafkaProducer - Closing the Kafka producer with timeoutMillis = 9223372036854775807 ms.
1001 [main] INFO org.apache.kafka.clients.producer.KafkaProducer - Closing the Kafka producer with timeoutMillis = 9223372036854775807 ms.

Producing job completed
```

3. Run Kafka Consumer:

- Run the command

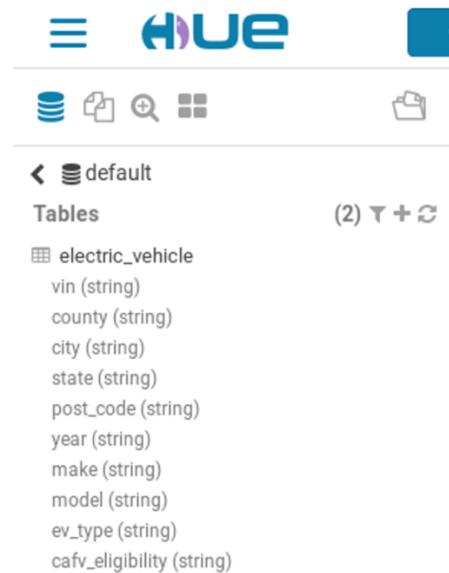
```
java -cp "target/FinalProject-0.0.1-SNAPSHOT.jar" cs523.miu.App
```



A terminal window titled "cloudera@quickstart:~/workspace/FinalProject" displaying the output of a Java application. The logs show various INFO messages from org.apache.spark components, including Executor, TaskSetManager, DAGScheduler, and JobScheduler, indicating the execution of tasks and removal of RDDs. The log ends with a message about removing old batch metadata.

```
cloudera@quickstart:~/workspace/FinalProject
File Edit View Search Terminal Help
104067 [Executor task launch worker for task 365] INFO org.apache.spark.executor.Executor - Finished task 0.0 in stage 365.0 (TID 365). 665 bytes result sent to driver
104067 [Executor task launch worker for task 365] INFO org.apache.spark.executor.Executor - Finished task 0.0 in stage 365.0 (TID 365). 665 bytes result sent to driver
104068 [TaskResultGetter-1] INFO org.apache.spark.scheduler.TaskSetManager - Finished task 0.0 in stage 365.0 (TID 365) in 4 ms on localhost (executor driver) (1/1)
104068 [TaskResultGetter-1] INFO org.apache.spark.scheduler.TaskSetManager - Finished task 0.0 in stage 365.0 (TID 365) in 4 ms on localhost (executor driver) (1/1)
104068 [TaskResultGetter-1] INFO org.apache.spark.scheduler.TaskSchedulerImpl - Removed TaskSet 365.0, whose tasks have all completed, from pool
104068 [TaskResultGetter-1] INFO org.apache.spark.scheduler.TaskSchedulerImpl - Removed TaskSet 365.0, whose tasks have all completed, from pool
104068 [dag-scheduler-event-loop] INFO org.apache.spark.scheduler.DAGScheduler - ResultStage 365 (foreach at KConsumer.java:48) finished in 0.005 s
104068 [dag-scheduler-event-loop] INFO org.apache.spark.scheduler.DAGScheduler - ResultStage 365 (foreach at KConsumer.java:48) finished in 0.005 s
104069 [streaming-job-executor-0] INFO org.apache.spark.scheduler.DAGScheduler - Job 365 finished: foreach at KConsumer.java:48, took 0.010208 s
104069 [streaming-job-executor-0] INFO org.apache.spark.scheduler.DAGScheduler - Job 365 finished: foreach at KConsumer.java:48, took 0.010208 s
104069 [JobScheduler] INFO org.apache.spark.streaming.scheduler.JobScheduler - Finished job streaming job 1671316541250 ms.0 from job set of time 1671316541250 ms
104069 [JobScheduler] INFO org.apache.spark.streaming.scheduler.JobScheduler - Finished job streaming job 1671316541250 ms.0 from job set of time 1671316541250 ms
104069 [JobScheduler] INFO org.apache.spark.streaming.scheduler.JobScheduler - Total delay: 0.021 s for time 1671316541250 ms (execution: 0.013 s)
104069 [JobScheduler] INFO org.apache.spark.streaming.scheduler.JobScheduler - Total delay: 0.021 s for time 1671316541250 ms (execution: 0.013 s)
104070 [JobGenerator] INFO org.apache.spark.streaming.kafka010.KafkaRDD - Removing RDD 364 from persistence list
104070 [JobGenerator] INFO org.apache.spark.streaming.kafka010.KafkaRDD - Removing RDD 364 from persistence list
104070 [block-manager-slave-async-thread-pool-1] INFO org.apache.spark.storage.BlockManager - Removing RDD 364
104070 [block-manager-slave-async-thread-pool-1] INFO org.apache.spark.storage.BlockManager - Removing RDD 364
104071 [JobGenerator] INFO org.apache.spark.streaming.scheduler.ReceivedBlockTracker - Deleting batches:
104071 [JobGenerator] INFO org.apache.spark.streaming.scheduler.ReceivedBlockTracker - Deleting batches:
104071 [JobGenerator] INFO org.apache.spark.streaming.scheduler.InputInfoTracker - remove old batch metadata: 1671316540750 ms
104071 [JobGenerator] INFO org.apache.spark.streaming.scheduler.InputInfoTracker - remove old batch metadata: 1671316540750 ms
```

- 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_vehicle.year
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:

- 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

```
File Edit View Search Terminal Help
9323 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/76098a55-b928-4a94-a474-d44920f4a895
9323 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/76098a55-b928-4a94-a474-d44920f4a895
9332 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/76098a55-b928-4a94-a474-d44920f4a895
9332 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/76098a55-b928-4a94-a474-d44920f4a895
9349 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/76098a55-b928-4a94-a474-d44920f4a895
9349 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/76098a55-b928-4a94-a474-d44920f4a895
9366 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/76098a55-b928-4a94-a474-d44920f4a895
9366 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/76098a55-b928-4a94-a474-d44920f4a895
9369 [main] INFO org.apache.spark.sql.hive.client.HiveClientImpl - Warehouse location for Hive client (version 1.2.1) is file:/home/cl
use
9369 [main] INFO org.apache.spark.sql.hive.client.HiveClientImpl - Warehouse location for Hive client (version 1.2.1) is file:/home/cl
se
9349 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/aa5dac9c-b74b-4c78-a608-f16c2b069642
9349 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/aa5dac9c-b74b-4c78-a608-f16c2b069642
9868 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/aa5dac9c-b74b-4c78-a608-f1
9868 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/aa5dac9c-b74b-4c78-a608-f1
9899 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/aa5dac9c-b74b-4c78-a608-f16c2
9899 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/aa5dac9c-b74b-4c78-a608-f16c2
9956 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/aa5dac9c-b74b-4c78-a608-f1
9956 [main] INFO org.apache.hadoop.hive.session.SessionState - Created local directory: /tmp/hive/cloudera/aa5dac9c-b74b-4c78-a608-f1
9956 [main] INFO org.apache.spark.sql.hive.client.HiveClientImpl - Warehouse location for Hive client (version 1.2.1) is file:/home/cl
use
9956 [main] INFO org.apache.spark.sql.hive.client.HiveClientImpl - Warehouse location for Hive client (version 1.2.1) is file:/home/cl
se
10144 [main] INFO org.apache.spark.sql.execution.streaming.state.StateStoreCoordinatorRef - Registered StateStoreCoordinator endpoint
10144 [main] INFO org.apache.spark.sql.execution.streaming.state.StateStoreCoordinatorRef - Registered StateStoreCoordinator endpoint
=====
Welcome to Electric Vehicle Application =====
Please select program:
1. Calculate the growth of EV in each year
2. Favourite EV model each year and its number
3. How many Plug-in Hybrid Electric Vehicles (PHEV), vs Battery Electric Vehicle (BEV)
Type "exit" to stop program.
```

- 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 analysis 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 analysis 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 analysis 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

- 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();
```

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

The screenshot shows an IDE interface with two tabs: *JavaSparkHiveExample.java* and *SparkSQL_Hive/pom.xml*. The *pom.xml* tab is active, displaying the following Maven configuration:

```
<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");

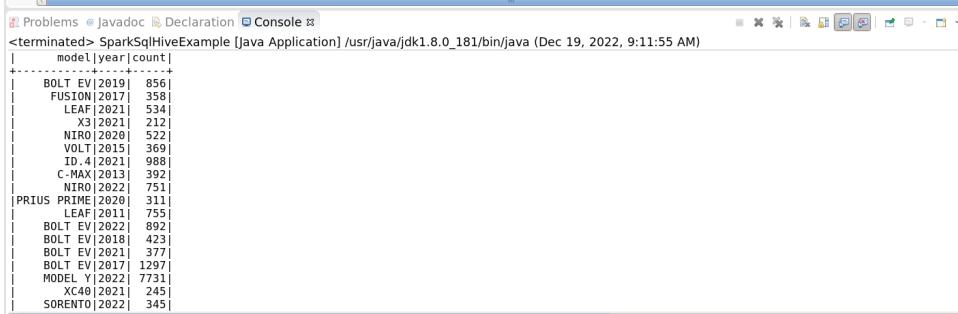
//chaining of dataset

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();
```



model	year	count
BOLT EV	2019	856
FUSION	2017	358
LEAF	2021	534
X3	2021	212
NIRO	2021	522
VOLT	2015	369
ID 4	2021	988
C-MAX	2013	392
NIRO	2022	751
PRIUS PRIME	2020	311
LEAF	2021	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

The screenshot shows the Hue Editor interface in Mozilla Firefox. The URL is `quickstart.cloudera:8888/hue/editor?editor=22`. The top navigation bar includes links for Cloudera, Hue, Hadoop, HBase, Impala, Spark, Solr, Oozie, Cloudera Manager, and Getting Started.

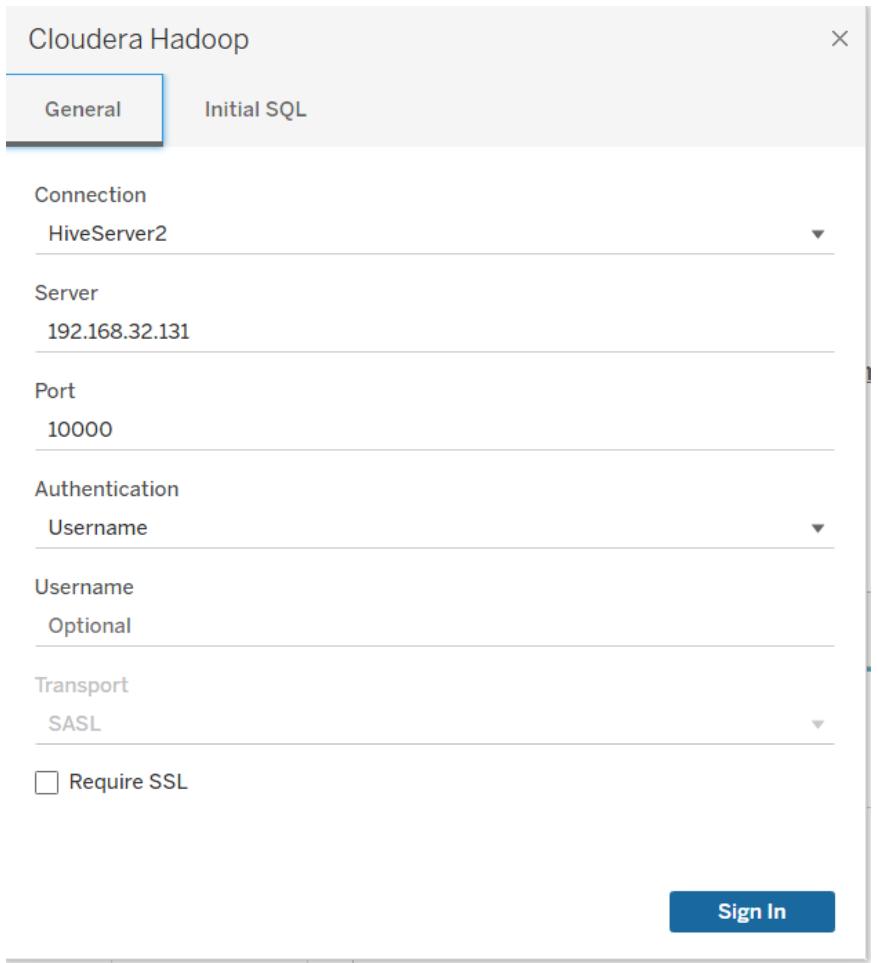
The main area has a left sidebar titled "Tables" under "default" which lists "countcar", "electricvehicles", "vehicletype", "model (string)", and "ev_type (string)". The central pane is titled "Hive" and contains a query editor with the following code:

```
1 select * from vehicletype
```

Below the query editor is a "Results (100+)" section. It displays two tables side-by-side:

	vehicletype.model	vehicletype.ev_type
1	Model	Electric Vehicle Type
2	RAV4 PRIME	Plug-in Hybrid Electric Vehicle (PHEV)
3	VOLT	Plug-in Hybrid Electric Vehicle (PHEV)
4	LEAF	Battery Electric Vehicle (BEV)
5	BOLT EV	Battery Electric Vehicle (BEV)
6	FUSION	Plug-in Hybrid Electric Vehicle (PHEV)
7	MODEL	Electric Vehicle Type

f. Connect tableau to the database



Connections

(92.168.32.131) Cloudera Hadoop

hema

default

ble

electricVehicles

Exact Contains Starts with

countcar (default countcar)

New Custom SQL

New Union

New Table Extension

electricVehicles (default.electricVehicles) (default)

electricVehicles

Need more data?
Drag tables here to relate them. [Learn more](#)

Name	Type	Field Name	Physical Table	Remote Field ...
Vin	Abc	Vin	electricVehicles	vin
County	⊕	County	electricVehicles	county
City	⊕	City	electricVehicles	city
State	⊕		electricVehicles	Post Code
Post Code	#		electricVehicles	Year

Update Now

Update Automatically

g. Visualize trends and analysis

