Logic For First Submission

<Properly explain the code, list the steps to run the code provided by you and attach screenshots of code execution>

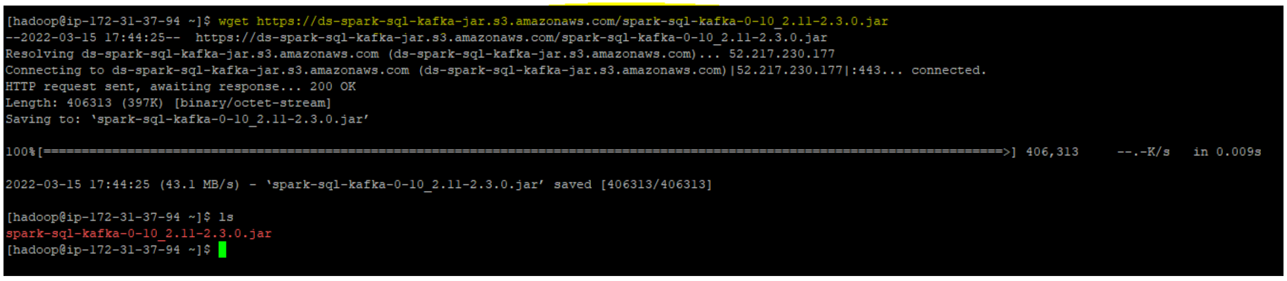
Note: Be as descriptive as possible.

Create an EMR cluster with following applications installed.

1. Hive 2.3.6
2. Spark 2.4.5
3. Sqoop 1.4.7
4. JupyterHub 1.1.0
5. Livy 0.7.0

- Now, run the below command to download the Spark-SQL-Kafka jar file. This jar will be used to run the Spark Streaming-Kafka codes.

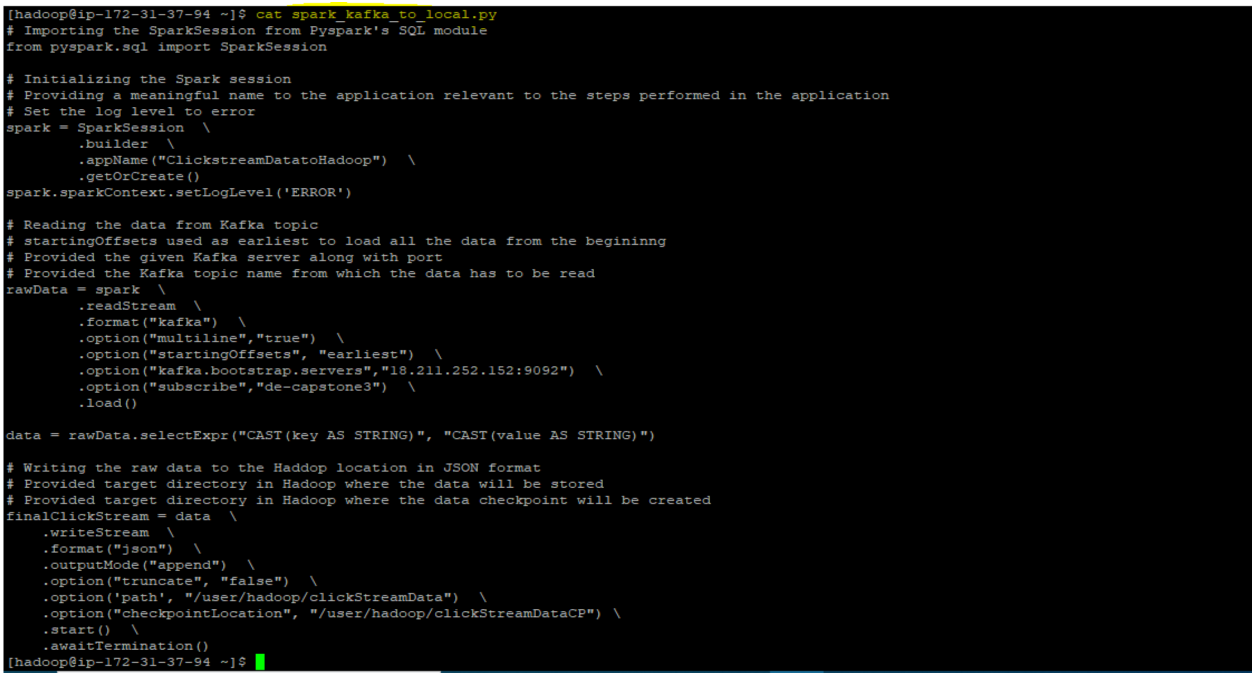
Command:
wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar



```
[hadoop@ip-172-31-37-94 ~]$ wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar
--2022-03-15 17:44:25--  https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar
Resolving ds-spark-sql-kafka-jar.s3.amazonaws.com (ds-spark-sql-kafka-jar.s3.amazonaws.com)... 52.217.230.177
Connecting to ds-spark-sql-kafka-jar.s3.amazonaws.com (ds-spark-sql-kafka-jar.s3.amazonaws.com)|52.217.230.177|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 406313 (397K) [binary/octet-stream]
Saving to: 'spark-sql-kafka-0-10_2.11-2.3.0.jar'

100%[=================================================================================================================>] 406,313     --.-K/s   in 0.009s

2022-03-15 17:44:25 (43.1 MB/s) - 'spark-sql-kafka-0-10_2.11-2.3.0.jar' saved [406313/406313]

[hadoop@ip-172-31-37-94 ~]$ ls
spark-sql-kafka-0-10_2.11-2.3.0.jar
[hadoop@ip-172-31-37-94 ~]$
```

1. Steps to run the python file to load data from Kafka

- A python file (**spark_kafka_to_local.py**) is created to ingest the relevant data from the given Kafka topic and load the data into hadoop file system. Below is the screenshot of the **spark_kafka_to_local.py** python file.



```
[hadoop@ip-172-31-37-94 ~]$ cat spark_kafka_to_local.py
# Importing the SparkSession from Pyspark's SQL module
from pyspark.sql import SparkSession

# Initializing the Spark session
# Providing a meaningful name to the application relevant to the steps performed in the application
# Set the log level to error
spark = SparkSession  \
        .builder  \
        .appName("ClickstreamDatatoHadoop")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Reading the data from Kafka topic
# startingOffsets used as earliest to load all the data from the begininng
# Provided the given Kafka server along with port
# Provided the Kafka topic name from which the data has to be read
rawData = spark  \
        .readStream  \
        .format("kafka")  \
        .option("multiline","true")  \
        .option("startingOffsets", "earliest")  \
        .option("kafka.bootstrap.servers","18.211.252.152:9092")  \
        .option("subscribe","de-capstone3")  \
        .load()

data = rawData.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")

# Writing the raw data to the Haddop location in JSON format
# Provided target directory in Hadoop where the data will be stored
# Provided target directory in Hadoop where the data checkpoint will be created
finalClickStream = data  \
    .writeStream  \
    .format("json")  \
    .outputMode("append")  \
    .option("truncate", "false")  \
    .option('path', "/user/hadoop/clickStreamData")  \
    .option("checkpointLocation", "/user/hadoop/clickStreamDataCP") \
    .start()  \
    .awaitTermination()
[hadoop@ip-172-31-37-94 ~]$
```

- Below is the spark-submit command to run the **spark_kafka_to_local.py** file.

  Command:
  spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark_kafka_to_local.py ##.###.###.### #### de-capstone3

- Data imported to hadoop can be listed using the following command.

  Command:
  hadoop fs –ls /user/hadoop/clickStreamData

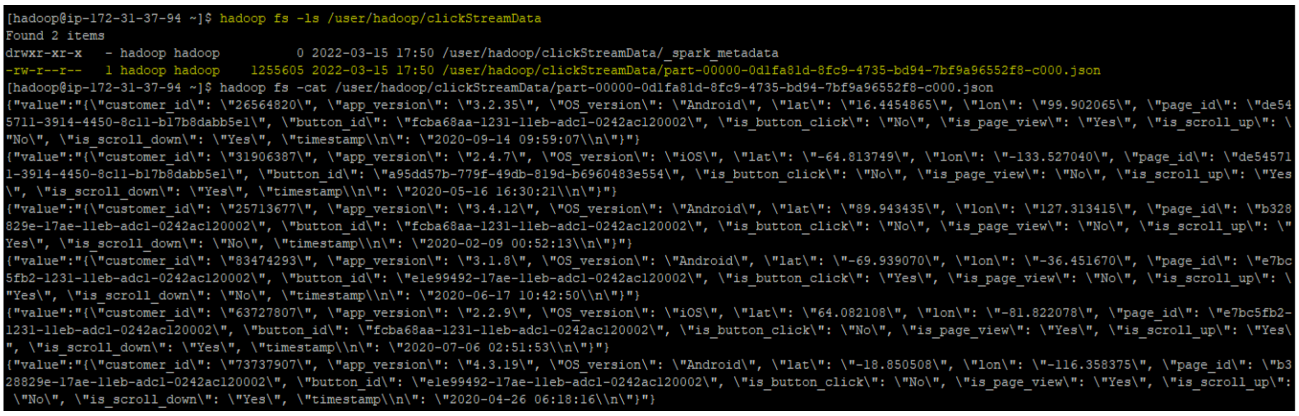
- Data from the JSON file can be viewed using the following command.

  Command:
  hadoop fs –cat /user/hadoop/clickStreamData/<filename>

  OR

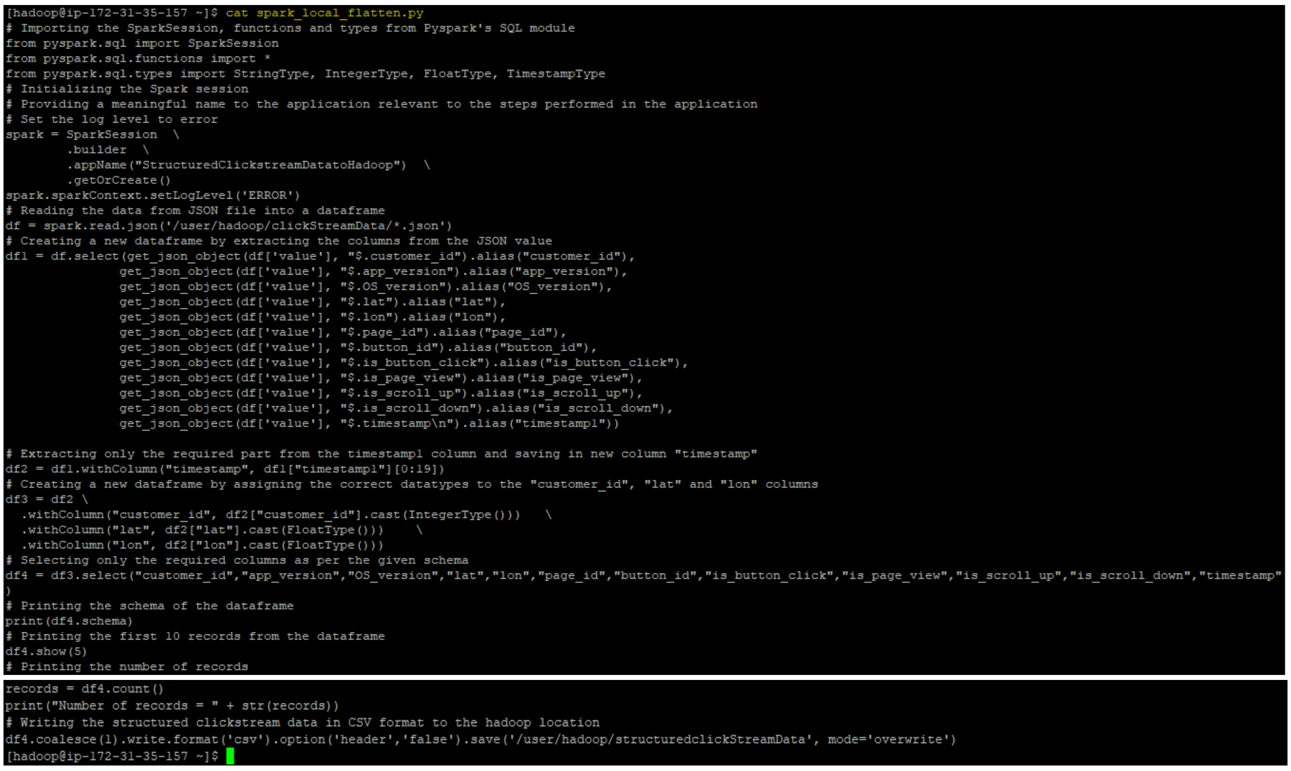
  hadoop fs –cat /user/hadoop/clickStreamData/*.json

- Below is the screenshot of the data imported from Kafka to Hadoop.



```
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -ls /user/hadoop/clickStreamData
Found 2 items
drwxr-xr-x   - hadoop hadoop          0 2022-03-15 17:50 /user/hadoop/clickStreamData/_spark_metadata
-rw-r--r--   1 hadoop hadoop    1255605 2022-03-15 17:50 /user/hadoop/clickStreamData/part-00000-0d1fa81d-8fc9-4735-bd94-7bf9a96552f8-c000.json
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -cat /user/hadoop/clickStreamData/part-00000-0d1fa81d-8fc9-4735-bd94-7bf9a96552f8-c000.json
{"value":"{\"customer_id\": \"26564820\", \"app_version\": \"3.2.35\", \"OS_version\": \"Android\", \"lat\": \"16.4454865\", \"lon\": \"99.902065\", \"page_id\": \"de54
5711-3914-4450-8c11-b17b8dabb5e1\", \"button_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\": \
"No\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-09-14 09:59:07\\n\"}"}
{"value":"{\"customer_id\": \"31906387\", \"app_version\": \"2.4.7\", \"OS_version\": \"iOS\", \"lat\": \"-64.813749\", \"lon\": \"-133.527040\", \"page_id\": \"de54571
1-3914-4450-8c11-b17b8dabb5e1\", \"button_id\": \"a95dd57b-779f-49db-819d-b6960483e554\", \"is_button_click\": \"No\", \"is_page_view\": \"No\", \"is_scroll_up\": \"Yes
\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-05-16 16:30:21\\n\"}"}
{"value":"{\"customer_id\": \"25713677\", \"app_version\": \"3.4.12\", \"OS_version\": \"Android\", \"lat\": \"89.943435\", \"lon\": \"127.313415\", \"page_id\": \"b328
829e-17ae-11eb-adc1-0242ac120002\", \"button_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"No\", \"is_scroll_up\": \"
Yes\", \"is_scroll_down\": \"No\", \"timestamp\\n\": \"2020-02-09 00:52:13\\n\"}"}
{"value":"{\"customer_id\": \"83474293\", \"app_version\": \"3.1.8\", \"OS_version\": \"Android\", \"lat\": \"-69.939070\", \"lon\": \"-36.451670\", \"page_id\": \"e7bc
5fb2-1231-11eb-adc1-0242ac120002\", \"button_id\": \"e1e99492-17ae-11eb-adc1-0242ac120002\", \"is_button_click\": \"Yes\", \"is_page_view\": \"No\", \"is_scroll_up\": \
"Yes\", \"is_scroll_down\": \"No\", \"timestamp\\n\": \"2020-06-17 10:42:50\\n\"}"}
{"value":"{\"customer_id\": \"63727807\", \"app_version\": \"2.2.9\", \"OS_version\": \"iOS\", \"lat\": \"64.082108\", \"lon\": \"-81.822078\", \"page_id\": \"e7bc5fb2-
1231-11eb-adc1-0242ac120002\", \"button_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\": \"Yes\
", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-07-06 02:51:53\\n\"}"}
{"value":"{\"customer_id\": \"73737907\", \"app_version\": \"4.3.19\", \"OS_version\": \"Android\", \"lat\": \"-18.850508\", \"lon\": \"-116.358375\", \"page_id\": \"b3
28829e-17ae-11eb-adc1-0242ac120002\", \"button_id\": \"e1e99492-17ae-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\":
 \"No\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-04-26 06:18:16\\n\"}"}
```

- Another python file (**spark_local_flatten.py**) is created to clean the loaded Kafka data to a more structured format. Now we will run this python file. Below is the screenshot of the **spark_local_flatten.py** python file.

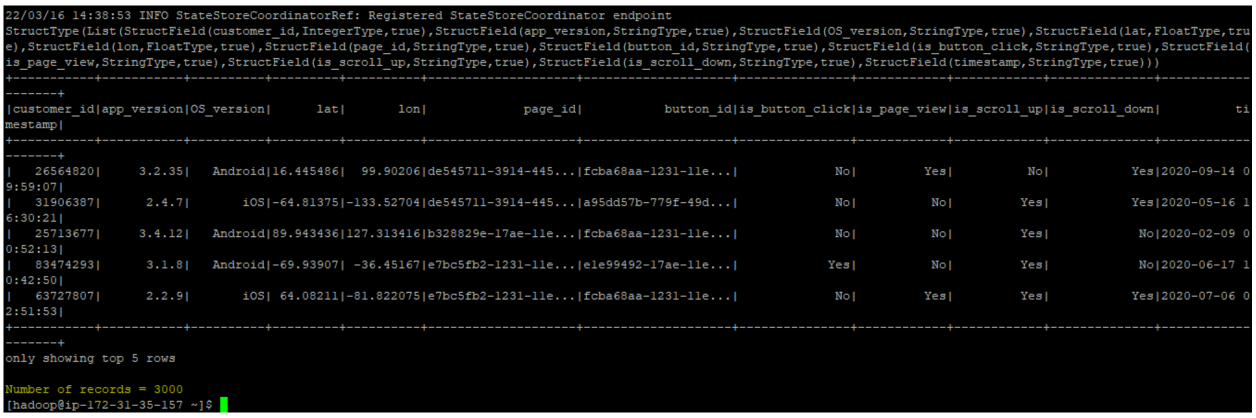


```
[hadoop@ip-172-31-35-157 ~]$ cat spark_local_flatten.py
# Importing the SparkSession, functions and types from Pyspark's SQL module
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StringType, IntegerType, FloatType, TimestampType
# Initializing the Spark session
# Providing a meaningful name to the application relevant to the steps performed in the application
# Set the log level to error
spark = SparkSession  \
        .builder  \
        .appName("StructuredClickstreamDatatoHadoop")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
# Reading the data from JSON file into a dataframe
df = spark.read.json('/user/hadoop/clickStreamData/*.json')
# Creating a new dataframe by extracting the columns from the JSON value
df1 = df.select(get_json_object(df['value'], "$.customer_id").alias("customer_id"),
               get_json_object(df['value'], "$.app_version").alias("app_version"),
               get_json_object(df['value'], "$.OS_version").alias("OS_version"),
               get_json_object(df['value'], "$.lat").alias("lat"),
               get_json_object(df['value'], "$.lon").alias("lon"),
               get_json_object(df['value'], "$.page_id").alias("page_id"),
               get_json_object(df['value'], "$.button_id").alias("button_id"),
               get_json_object(df['value'], "$.is_button_click").alias("is_button_click"),
               get_json_object(df['value'], "$.is_page_view").alias("is_page_view"),
               get_json_object(df['value'], "$.is_scroll_up").alias("is_scroll_up"),
               get_json_object(df['value'], "$.is_scroll_down").alias("is_scroll_down"),
               get_json_object(df['value'], "$.timestamp\n").alias("timestamp1"))

# Extracting only the required part from the timestamp1 column and saving in new column "timestamp"
df2 = df1.withColumn("timestamp", df1["timestamp1"][0:19])
# Creating a new dataframe by assigning the correct datatypes to the "customer_id", "lat" and "lon" columns
df3 = df2 \
  .withColumn("customer_id", df2["customer_id"].cast(IntegerType()))   \
  .withColumn("lat", df2["lat"].cast(FloatType()))    \
  .withColumn("lon", df2["lon"].cast(FloatType()))
# Selecting only the required columns as per the given schema
df4 = df3.select("customer_id","app_version","OS_version","lat","lon","page_id","button_id","is_button_click","is_page_view","is_scroll_up","is_scroll_down","timestamp"
)
# Printing the schema of the dataframe
print(df4.schema)
# Printing the first 10 records from the dataframe
df4.show(5)
# Printing the number of records
records = df4.count()
print("Number of records = " + str(records))
# Writing the structured clickstream data in CSV format to the hadoop location
df4.coalesce(1).write.format('csv').option('header','false').save('/user/hadoop/structuredclickStreamData', mode='overwrite')
[hadoop@ip-172-31-35-157 ~]$
```

- Below is the spark-submit command to run the **spark_local_flatten.py** file.

  Command:
  spark-submit spark_local_flatten.py

- Below is the screenshot of the output after running the **spark_local_flatten.py file.**



```
22/03/16 14:38:53 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint
StructType(List(StructField(customer_id,IntegerType,true),StructField(app_version,StringType,true),StructField(OS_version,StringType,true),StructField(lat,FloatType,tru
e),StructField(lon,FloatType,true),StructField(page_id,StringType,true),StructField(button_id,StringType,true),StructField(is_button_click,StringType,true),StructField(
is_page_view,StringType,true),StructField(is_scroll_up,StringType,true),StructField(is_scroll_down,StringType,true),StructField(timestamp,StringType,true)))
+-----------+-----------+----------+---------+----------+--------------------+--------------------+---------------+------------+------------+--------------+------------
-------+
|customer_id|app_version|OS_version|      lat|       lon|             page_id|           button_id|is_button_click|is_page_view|is_scroll_up|is_scroll_down|          ti
mestamp|
+-----------+-----------+----------+---------+----------+--------------------+--------------------+---------------+------------+------------+--------------+------------
-------+
|   26564820|     3.2.35|   Android|16.445486|  99.90206|de545711-3914-445...|fcba68aa-1231-11e...|             No|         Yes|          No|           Yes|2020-09-14 0
9:59:07|
|   31906387|      2.4.7|       iOS|-64.81375|-133.52704|de545711-3914-445...|a95dd57b-779f-49d...|             No|          No|         Yes|           Yes|2020-05-16 1
6:30:21|
|   25713677|     3.4.12|   Android|89.943436|127.313416|b328829e-17ae-11e...|fcba68aa-1231-11e...|             No|          No|         Yes|            No|2020-02-09 0
0:52:13|
|   83474293|      3.1.8|   Android|-69.93907| -36.45167|e7bc5fb2-1231-11e...|e1e99492-17ae-11e...|            Yes|          No|         Yes|            No|2020-06-17 1
0:42:50|
|   63727807|      2.2.9|       iOS| 64.08211|-81.822075|e7bc5fb2-1231-11e...|fcba68aa-1231-11e...|             No|         Yes|         Yes|           Yes|2020-07-06 0
2:51:53|
+-----------+-----------+----------+---------+----------+--------------------+--------------------+---------------+------------+------------+--------------+------------
-------+
only showing top 5 rows

Number of records = 3000
[hadoop@ip-172-31-35-157 ~]$
```

- Data is loaded into HDFS at **“/user/hadoop/structuredclickStreamData”** directory using the below command which is included in the **spark_local_flatten.py file.**

  Command:
  df4.coalesce(1).write.format('csv').option('header','false').save('/user/hadoop/structuredclickStreamData', mode='overwrite')

- Cleaned and structured data imported to hadoop after running the spark_local_flatten.py python file can be listed using the following command.

  Command:
  hadoop fs –ls /user/hadoop/structuredclickStreamData

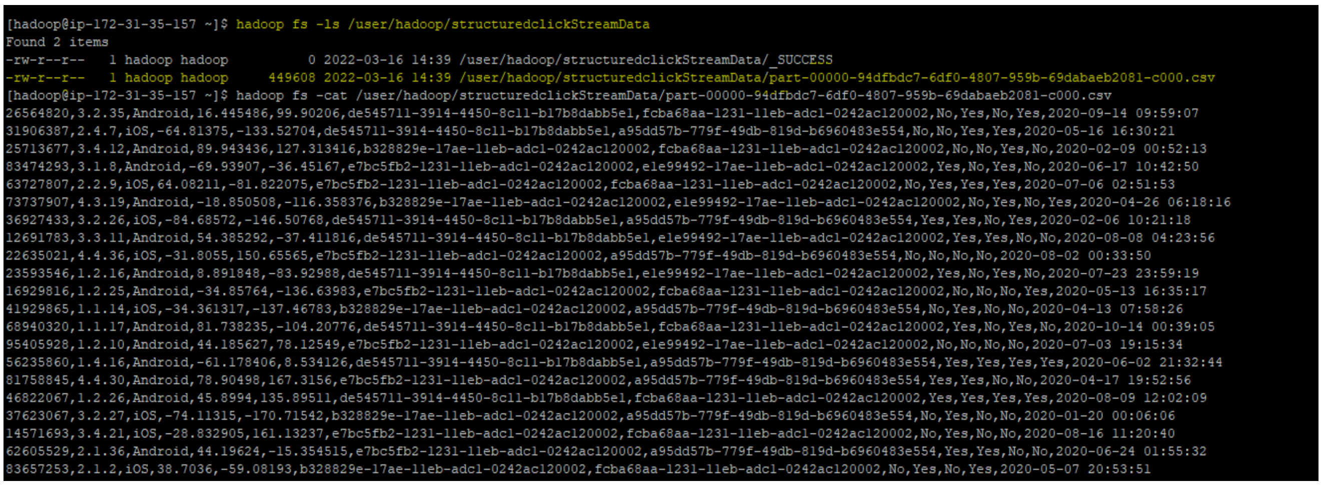
- Data from the CSV file can be viewed using the following command.

  Command:
  hadoop fs –cat /user/hadoop/structuredclickStreamData/<filename>

  OR

  hadoop fs –cat /user/hadoop/structuredclickStreamData/*.csv

- Below is the screenshot of the cleaned and structured data loaded in the Hadoop in CSV format. **(/user/hadoop/structuredclickStreamData)**



```
[hadoop@ip-172-31-35-157 ~]$ hadoop fs -ls /user/hadoop/structuredclickStreamData
Found 2 items
-rw-r--r--   1 hadoop hadoop          0 2022-03-16 14:39 /user/hadoop/structuredclickStreamData/_SUCCESS
-rw-r--r--   1 hadoop hadoop     449608 2022-03-16 14:39 /user/hadoop/structuredclickStreamData/part-00000-94dfbdc7-6df0-4807-959b-69dabaeb2081-c000.csv
[hadoop@ip-172-31-35-157 ~]$ hadoop fs -cat /user/hadoop/structuredclickStreamData/part-00000-94dfbdc7-6df0-4807-959b-69dabaeb2081-c000.csv
26564820,3.2.35,Android,16.445486,99.90206,de545711-3914-4450-8c11-b17b8dabb5e1,fcba68aa-1231-11eb-adc1-0242ac120002,No,Yes,No,Yes,2020-09-14 09:59:07
31906387,2.4.7,iOS,-64.81375,-133.52704,de545711-3914-4450-8c11-b17b8dabb5e1,a95dd57b-779f-49db-819d-b6960483e554,No,No,Yes,Yes,2020-05-16 16:30:21
25713677,3.4.12,Android,89.943436,127.313416,b328829e-17ae-11eb-adc1-0242ac120002,fcba68aa-1231-11eb-adc1-0242ac120002,No,No,Yes,No,2020-02-09 00:52:13
83474293,3.1.8,Android,-69.93907,-36.45167,e7bc5fb2-1231-11eb-adc1-0242ac120002,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,No,Yes,No,2020-06-17 10:42:50
63727807,2.2.9,iOS,64.08211,-81.822075,e7bc5fb2-1231-11eb-adc1-0242ac120002,fcba68aa-1231-11eb-adc1-0242ac120002,No,Yes,Yes,Yes,2020-07-06 02:51:53
73737907,4.3.19,Android,-18.850508,-116.358376,b328829e-17ae-11eb-adc1-0242ac120002,e1e99492-17ae-11eb-adc1-0242ac120002,No,Yes,No,Yes,2020-04-26 06:18:16
36927433,3.2.26,iOS,-84.68572,-146.50768,de545711-3914-4450-8c11-b17b8dabb5e1,a95dd57b-779f-49db-819d-b6960483e554,Yes,Yes,No,Yes,2020-02-06 10:21:18
12691783,3.3.11,Android,54.385292,-37.411816,de545711-3914-4450-8c11-b17b8dabb5e1,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,Yes,No,No,2020-08-08 04:23:56
22635021,4.4.36,iOS,-31.8055,150.65565,e7bc5fb2-1231-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,No,No,No,No,2020-08-02 00:33:50
23593546,1.2.16,Android,8.891848,-83.92988,de545711-3914-4450-8c11-b17b8dabb5e1,e1e99492-17ae-11eb-adc1-0242ac120002,Yes,No,Yes,No,2020-07-23 23:59:19
16929816,1.2.25,Android,-34.85764,-136.63983,e7bc5fb2-1231-11eb-adc1-0242ac120002,fcba68aa-1231-11eb-adc1-0242ac120002,No,No,No,Yes,2020-05-13 16:35:17
41929865,1.1.14,iOS,-34.361317,-137.46783,b328829e-17ae-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,No,Yes,No,No,2020-04-13 07:58:26
68940320,1.1.17,Android,81.738235,-104.20776,de545711-3914-4450-8c11-b17b8dabb5e1,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,No,Yes,No,2020-10-14 00:39:05
95405928,1.2.10,Android,44.185627,78.12549,e7bc5fb2-1231-11eb-adc1-0242ac120002,e1e99492-17ae-11eb-adc1-0242ac120002,No,No,No,No,2020-07-03 19:15:34
56235860,1.4.16,Android,-61.178406,8.534126,de545711-3914-4450-8c11-b17b8dabb5e1,a95dd57b-779f-49db-819d-b6960483e554,Yes,Yes,Yes,Yes,2020-06-02 21:32:44
81758845,4.4.30,Android,78.90498,167.3156,e7bc5fb2-1231-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,Yes,Yes,No,No,2020-04-17 19:52:56
46822067,1.2.26,Android,45.8994,135.89511,de545711-3914-4450-8c11-b17b8dabb5e1,fcba68aa-1231-11eb-adc1-0242ac120002,Yes,Yes,Yes,Yes,2020-08-09 12:02:09
37623067,3.2.27,iOS,-74.11315,-170.71542,b328829e-17ae-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,No,Yes,No,No,2020-01-20 00:06:06
14571693,3.4.21,iOS,-28.832905,161.13237,e7bc5fb2-1231-11eb-adc1-0242ac120002,fcba68aa-1231-11eb-adc1-0242ac120002,No,Yes,No,No,2020-08-16 11:20:40
62605529,2.1.36,Android,44.19624,-15.354515,e7bc5fb2-1231-11eb-adc1-0242ac120002,a95dd57b-779f-49db-819d-b6960483e554,Yes,Yes,No,No,2020-06-24 01:55:32
83657253,2.1.2,iOS,38.7036,-59.08193,b328829e-17ae-11eb-adc1-0242ac120002,fcba68aa-1231-11eb-adc1-0242ac120002,No,Yes,No,Yes,2020-05-07 20:53:51
```

2. Steps to run the Sqoop Command to import data from AWS RDS to Hadoop

We need to install MySQL connector before running the Sqoop import command.

We can install MySQL connector using the following steps.

1. Run the below command to install the MySQL connector jar file.
   Command:
   wget https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz

2. Run the below command to extract the MySQL connector tar file.

   Command:
   tar -xvf mysql-connector-java-8.0.25.tar.gz

3. Go to the MySQL Connector directory created in the previous step and then copy it to the Sqoop library to complete the installation. Run below commands.

   Commands:
   cd mysql-connector-java-8.0.25/
   sudo cp mysql-connector-java-8.0.25.jar /usr/lib/sqoop/lib/

4. Run the following command to start setting up MySQL.

   Command:
   mysql_secure_installation

5. The command will ask for the current password for root. We need to press Enter as no password is currently set for root.

6. Then, we will be asked to set the root password. We need to type **Y** and then press **Enter**.

7. Type the password we want to set then press enter. Re-enter the password to confirm and press enter.

8. We will be asked to remove anonymous users. Type **Y** and then press **Enter**.

9. We will be asked to disallow the root login remotely. Type **n** and then press **Enter**.

10. We will be asked to remove test database along with its access. Type **Y** and then press **Enter**.

11. We will be asked to reload privilege tables. Type **Y** and then press **Enter**.

12. **With this MySQL setup is complete. Give following command to enter the MySQL shell and enter the password.**

    Command:
    ```
    mysql -u root –p
    ```

13. Run the following queries to grant all privileges to the root user.

    Commands:

    ```
    GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' identified by '123' WITH GRANT OPTION;

    flush privileges;

    exit;
    ```

14. Run the following command to restart the MySQL service to finish setting up MySQL.

    Command:
    ```
    sudo service mariadb restart
    ```

MySQL installation is completed.

- Now, run the below sqoop command to import the bookings data from AWS RDS to the HDFS.

Command:
sqoop import \
--connect jdbc:mysql://######.######.us-east-1.rds.amazonaws.com/###### \
--table bookings \
--username student \
--password ###### \
--target-dir /user/hadoop/bookings_data \
-m 1

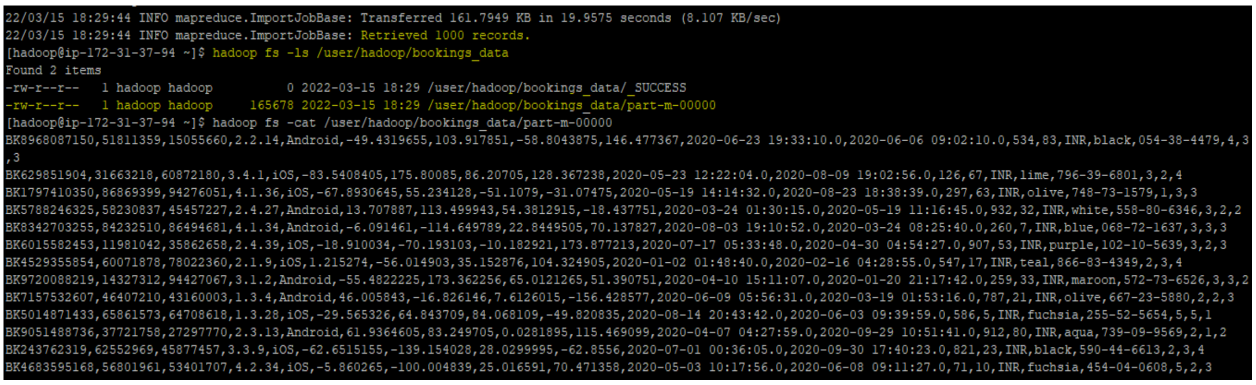
- Bookings data imported in the HDFS can be listed using the following command. The command will show two files named **_SUCCESS** and **part-m-00000** in the output.

  Command:
  hadoop fs -ls /user/hadoop/bookings_data

- Data from the file can be viewed using the following command.

  Command:
  hadoop fs –cat /user/hadoop/bookings_data/<filename>

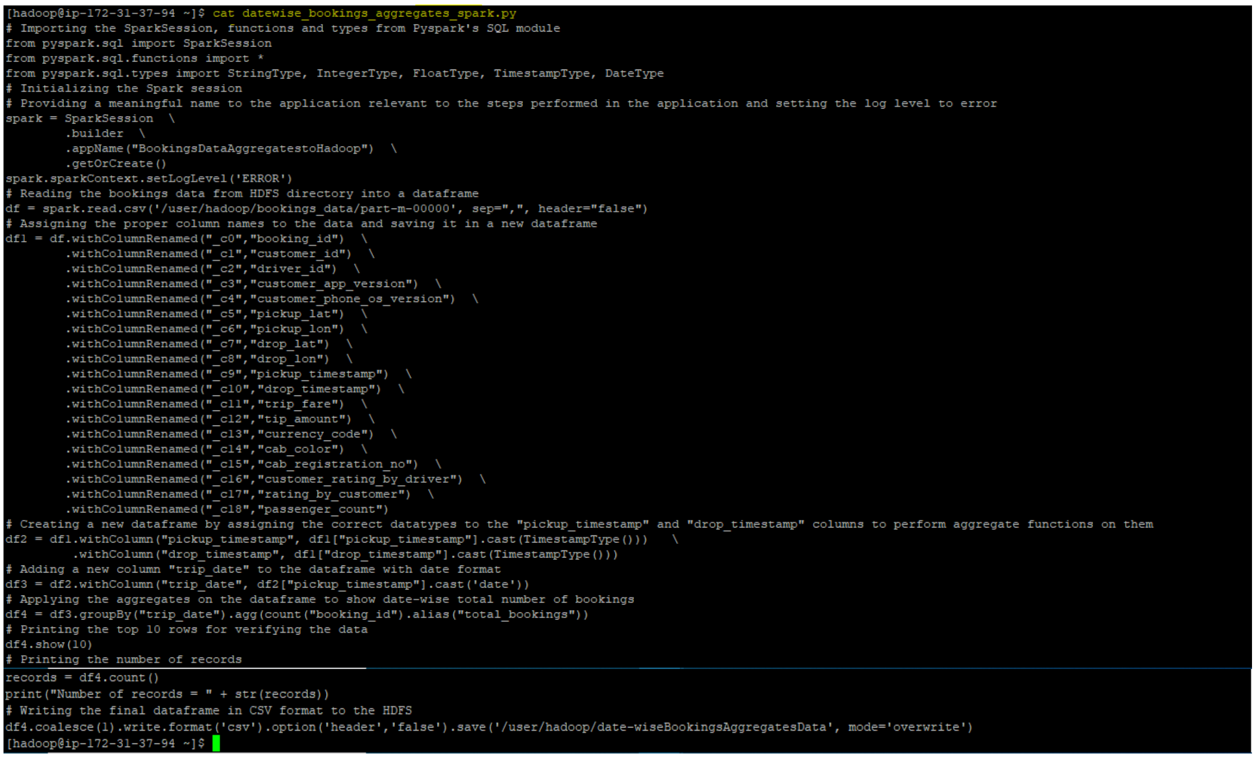
- Below is the screenshot of the data imported from AWS RDS to HDFS.



```
22/03/15 18:29:44 INFO mapreduce.ImportJobBase: Transferred 161.7949 KB in 19.9575 seconds (8.107 KB/sec)
22/03/15 18:29:44 INFO mapreduce.ImportJobBase: Retrieved 1000 records.
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -ls /user/hadoop/bookings_data
Found 2 items
-rw-r--r--   1 hadoop hadoop          0 2022-03-15 18:29 /user/hadoop/bookings_data/_SUCCESS
-rw-r--r--   1 hadoop hadoop     165678 2022-03-15 18:29 /user/hadoop/bookings_data/part-m-00000
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -cat /user/hadoop/bookings_data/part-m-00000
BK8968087150,51811359,15055660,2.2.14,Android,-49.4319655,103.917851,-58.8043875,146.477367,2020-06-23 19:33:10.0,2020-06-06 09:02:10.0,534,83,INR,black,054-38-4479,4,3
,3
BK629851904,31663218,60872180,3.4.1,iOS,-83.5408405,175.80085,86.20705,128.367238,2020-05-23 12:22:04.0,2020-08-09 19:02:56.0,126,67,INR,lime,796-39-6801,3,2,4
BK1797410350,86869399,94276051,4.1.36,iOS,-67.8930645,55.234128,-51.1079,-31.07475,2020-05-19 14:14:32.0,2020-08-23 18:38:39.0,297,63,INR,olive,748-73-1579,1,3,3
BK5788246325,58230837,45457227,2.4.27,Android,13.707887,113.499943,54.3812915,-18.437751,2020-03-24 01:30:15.0,2020-05-19 11:16:45.0,932,32,INR,white,558-80-6346,3,2,2
BK8342703255,84232510,86494681,4.1.34,Android,-6.091461,-114.649789,22.8449505,70.137827,2020-08-03 19:10:52.0,2020-03-24 08:25:40.0,260,7,INR,blue,068-72-1637,3,3,3
BK6015582453,11981042,35862658,2.4.39,iOS,-18.910034,-70.193103,-10.182921,173.877213,2020-07-17 05:33:48.0,2020-04-30 04:54:27.0,907,53,INR,purple,102-10-5639,3,2,3
BK4529355854,60071878,78022360,2.1.9,iOS,1.215274,-56.014903,35.152876,104.324905,2020-01-02 01:48:40.0,2020-02-16 04:28:55.0,547,17,INR,teal,866-83-4349,2,3,4
BK9720088219,14327312,94427067,3.1.2,Android,-55.4822225,173.362256,65.0121265,51.390751,2020-04-10 15:11:07.0,2020-01-20 21:17:42.0,259,33,INR,maroon,572-73-6526,3,3,2
BK7157532607,46407210,43160003,1.3.4,Android,46.005843,-16.826146,7.6126015,-156.428577,2020-06-09 05:56:31.0,2020-03-19 01:53:16.0,787,21,INR,olive,667-23-5880,2,2,3
BK5014871433,65861573,64708618,1.3.28,iOS,-29.565326,64.843709,84.068109,-49.820835,2020-08-14 20:43:42.0,2020-06-03 09:39:59.0,586,5,INR,fuchsia,255-52-5654,5,5,1
BK9051488736,37721758,27297770,2.3.13,Android,61.9364605,83.249705,0.0281895,115.469099,2020-04-07 04:27:59.0,2020-09-29 10:51:41.0,912,80,INR,aqua,739-09-9569,2,1,2
BK243762319,62552969,45877457,3.3.9,iOS,-62.6515155,-139.154028,28.0299995,-62.8556,2020-07-01 00:36:05.0,2020-09-30 17:40:23.0,821,23,INR,black,590-44-6613,2,3,4
BK4683595168,56801961,53401707,4.2.34,iOS,-5.860265,-100.004839,25.016591,70.471358,2020-05-03 10:17:56.0,2020-06-08 09:11:27.0,71,10,INR,fuchsia,454-04-0608,5,2,3
```

3. Steps to create aggregates to find date-wise total bookings

- A python file (**datewise_bookings_aggregates_spark.py**) is created to create aggregates for finding date-wise total bookings, store the data in CSV format and move the data to HDFS. Below is the screenshot of the datewise_bookings_aggregates_spark.py python file.



```
[hadoop@ip-172-31-37-94 ~]$ cat datewise_bookings_aggregates_spark.py
# Importing the SparkSession, functions and types from Pyspark's SQL module
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StringType, IntegerType, FloatType, TimestampType, DateType
# Initializing the Spark session
# Providing a meaningful name to the application relevant to the steps performed in the application and setting the log level to error
spark = SparkSession  \
        .builder  \
        .appName("BookingsDataAggregatestoHadoop")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
# Reading the bookings data from HDFS directory into a dataframe
df = spark.read.csv('/user/hadoop/bookings_data/part-m-00000', sep=",", header="false")
# Assigning the proper column names to the data and saving it in a new dataframe
df1 = df.withColumnRenamed("_c0","booking_id")  \
        .withColumnRenamed("_c1","customer_id")  \
        .withColumnRenamed("_c2","driver_id")  \
        .withColumnRenamed("_c3","customer_app_version")  \
        .withColumnRenamed("_c4","customer_phone_os_version")  \
        .withColumnRenamed("_c5","pickup_lat")  \
        .withColumnRenamed("_c6","pickup_lon")  \
        .withColumnRenamed("_c7","drop_lat")  \
        .withColumnRenamed("_c8","drop_lon")  \
        .withColumnRenamed("_c9","pickup_timestamp")  \
        .withColumnRenamed("_c10","drop_timestamp")  \
        .withColumnRenamed("_c11","trip_fare")  \
        .withColumnRenamed("_c12","tip_amount")  \
        .withColumnRenamed("_c13","currency_code")  \
        .withColumnRenamed("_c14","cab_color")  \
        .withColumnRenamed("_c15","cab_registration_no")  \
        .withColumnRenamed("_c16","customer_rating_by_driver")  \
        .withColumnRenamed("_c17","rating_by_customer")  \
        .withColumnRenamed("_c18","passenger_count")
# Creating a new dataframe by assigning the correct datatypes to the "pickup_timestamp" and "drop_timestamp" columns to perform aggregate functions on them
df2 = df1.withColumn("pickup_timestamp", df1["pickup_timestamp"].cast(TimestampType()))   \
         .withColumn("drop_timestamp", df1["drop_timestamp"].cast(TimestampType()))
# Adding a new column "trip_date" to the dataframe with date format
df3 = df2.withColumn("trip_date", df2["pickup_timestamp"].cast('date'))
# Applying the aggregates on the dataframe to show date-wise total number of bookings
df4 = df3.groupBy("trip_date").agg(count("booking_id").alias("total_bookings"))
# Printing the top 10 rows for verifying the data
df4.show(10)
# Printing the number of records
records = df4.count()
print("Number of records = " + str(records))
# Writing the final dataframe in CSV format to the HDFS
df4.coalesce(1).write.format('csv').option('header','false').save('/user/hadoop/date-wiseBookingsAggregatesData', mode='overwrite')
[hadoop@ip-172-31-37-94 ~]$
```

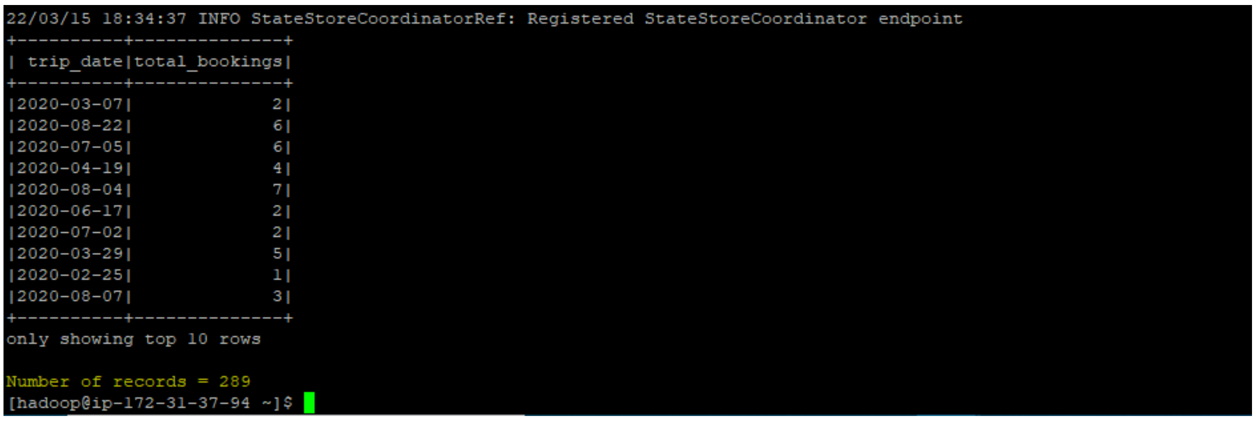
- Below is the spark-submit command to run **datewise_bookings_aggregates_spark.py file.**

  Command:
  spark-submit datewise_bookings_aggregates_spark.py

- Below is the command used to move the date-wise aggregates CSV file to the HDFS. (Below line of code is included in the **datewise_bookings_aggregates_spark.py** file).

  Command:
  df4.coalesce(1).write.format('csv').option('header','false').save('/user/hadoop/date-wiseBookingsAggregatesData', mode='overwrite')

- Below is the screenshot of the output after running the python file (**datewise_bookings_aggregates_spark.py**).



```
22/03/15 18:34:37 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint
+----------+--------------+
| trip_date|total_bookings|
+----------+--------------+
|2020-03-07|             2|
|2020-08-22|             6|
|2020-07-05|             6|
|2020-04-19|             4|
|2020-08-04|             7|
|2020-06-17|             2|
|2020-07-02|             2|
|2020-03-29|             5|
|2020-02-25|             1|
|2020-08-07|             3|
+----------+--------------+
only showing top 10 rows

Number of records = 289
[hadoop@ip-172-31-37-94 ~]$
```

- Date-wise total bookings aggregates data imported to hadoop after running the datewise_bookings_aggregates_spark.py python file can be listed using the following command.

  Command:
  hadoop fs –ls /user/hadoop/date-wiseBookingsAggregatesData

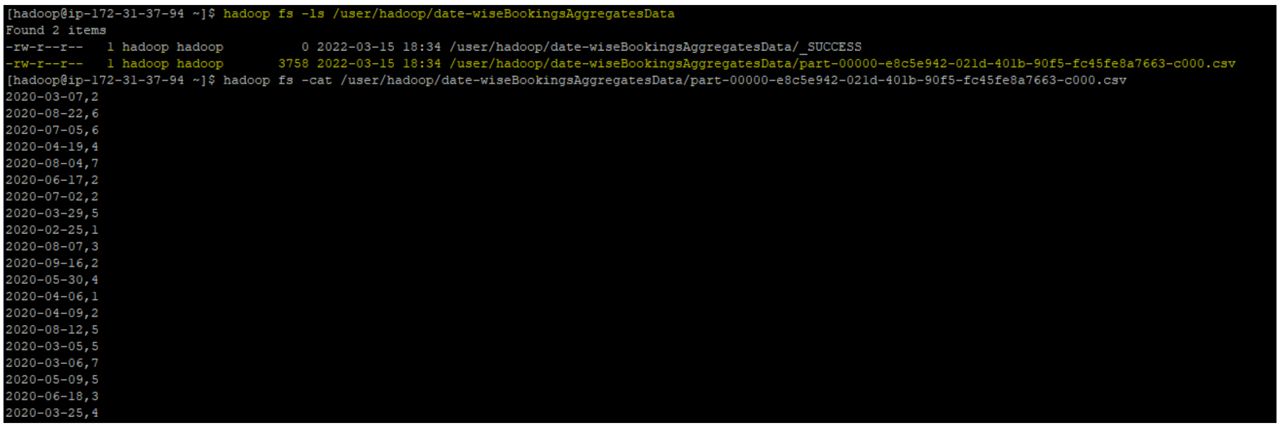
- Data from the CSV file can be viewed using the following command.

  Command:
  hadoop fs –cat /user/hadoop/date-wiseBookingsAggregatesData/<filename>

  OR

  hadoop fs –cat /user/hadoop/date-wiseBookingsAggregatesData/*.csv

- Below is the screenshot of the cleaned and structured data loaded in the Hadoop in CSV format. **(/user/hadoop/structuredclickStreamData)**



```
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -ls /user/hadoop/date-wiseBookingsAggregatesData
Found 2 items
-rw-r--r--   1 hadoop hadoop          0 2022-03-15 18:34 /user/hadoop/date-wiseBookingsAggregatesData/_SUCCESS
-rw-r--r--   1 hadoop hadoop       3758 2022-03-15 18:34 /user/hadoop/date-wiseBookingsAggregatesData/part-00000-e8c5e942-021d-401b-90f5-fc45fe8a7663-c000.csv
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -cat /user/hadoop/date-wiseBookingsAggregatesData/part-00000-e8c5e942-021d-401b-90f5-fc45fe8a7663-c000.csv
2020-03-07,2
2020-08-22,6
2020-07-05,6
2020-04-19,4
2020-08-04,7
2020-06-17,2
2020-07-02,2
2020-03-29,5
2020-02-25,1
2020-08-07,3
2020-09-16,2
2020-05-30,4
2020-04-06,1
2020-04-09,2
2020-08-12,5
2020-03-05,5
2020-03-06,7
2020-05-09,5
2020-06-18,3
2020-03-25,4
```

4. Steps to create Hive tables and load the data in Hive tables

- Give the command **“hive”** to launch the Hive CLI.

- Following command is used to create the database. Name of the database created is “**yourowncabs**”.

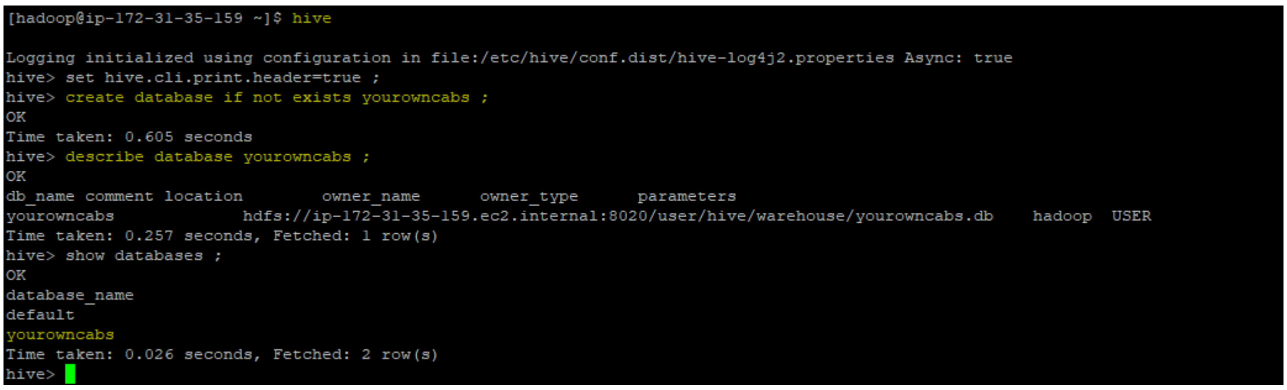
  Command:
  create database if not exists yourowncabs ;

- Following commands are used to list the details of the database created and to list the databases present.

  Commands:
  describe database yourowncabs ;

  show databases ;

- Below is the screenshot of the commands used to launch the Hive CLI and create the database.



```
[hadoop@ip-172-31-35-159 ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: true
hive> set hive.cli.print.header=true ;
hive> create database if not exists yourowncabs ;
OK
Time taken: 0.605 seconds
hive> describe database yourowncabs ;
OK
db_name comment location        owner_name      owner_type      parameters
yourowncabs             hdfs://ip-172-31-35-159.ec2.internal:8020/user/hive/warehouse/yourowncabs.db    hadoop  USER
Time taken: 0.257 seconds, Fetched: 1 row(s)
hive> show databases ;
OK
database_name
default
yourowncabs
Time taken: 0.026 seconds, Fetched: 2 row(s)
hive>
```

Below are the commands used to create the Hive tables.

- Command used to create Hive-managed table for **clickstream** data.

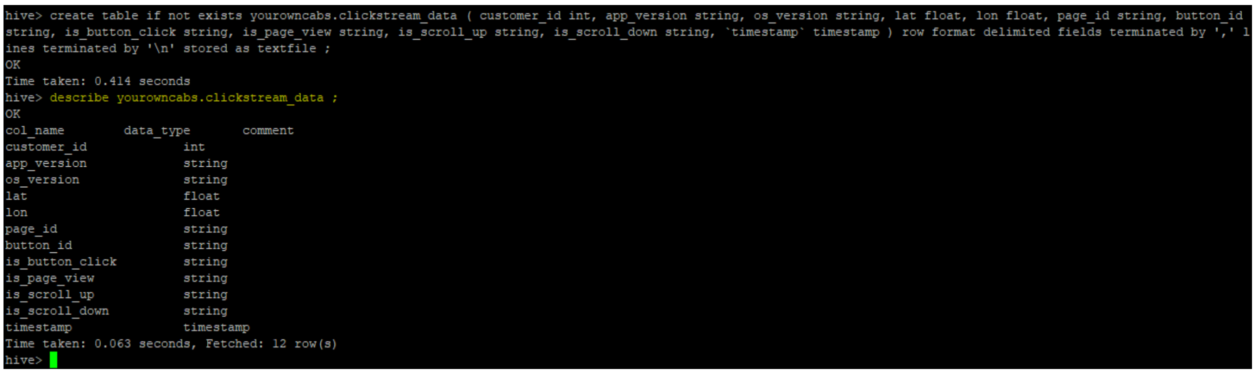
  Command:

  create table if not exists yourowncabs.clickstream_data (customer_id int, app_version string, os_version string, lat float, lon float, page_id string, button_id string, is_button_click string, is_page_view string, is_scroll_up string, is_scroll_down string, `timestamp` timestamp) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile ;

- Below is the command used to describe the clickstream_data table.

  Command:
  Describe yourowncabs.clickstream_data ;

- Below is the screenshot of the clickstream_data table created.



```
hive> create table if not exists yourowncabs.clickstream_data ( customer_id int, app_version string, os_version string, lat float, lon float, page_id string, button_id
string, is_button_click string, is_page_view string, is_scroll_up string, is_scroll_down string, `timestamp` timestamp ) row format delimited fields terminated by ',' l
ines terminated by '\n' stored as textfile ;
OK
Time taken: 0.414 seconds
hive> describe yourowncabs.clickstream_data ;
OK
col_name        data_type       comment
customer_id             int
app_version             string
os_version              string
lat                     float
lon                     float
page_id                 string
button_id               string
is_button_click         string
is_page_view            string
is_scroll_up            string
is_scroll_down          string
timestamp               timestamp
Time taken: 0.063 seconds, Fetched: 12 row(s)
hive>
```

- Command used to create Hive-managed table for **bookings** data.

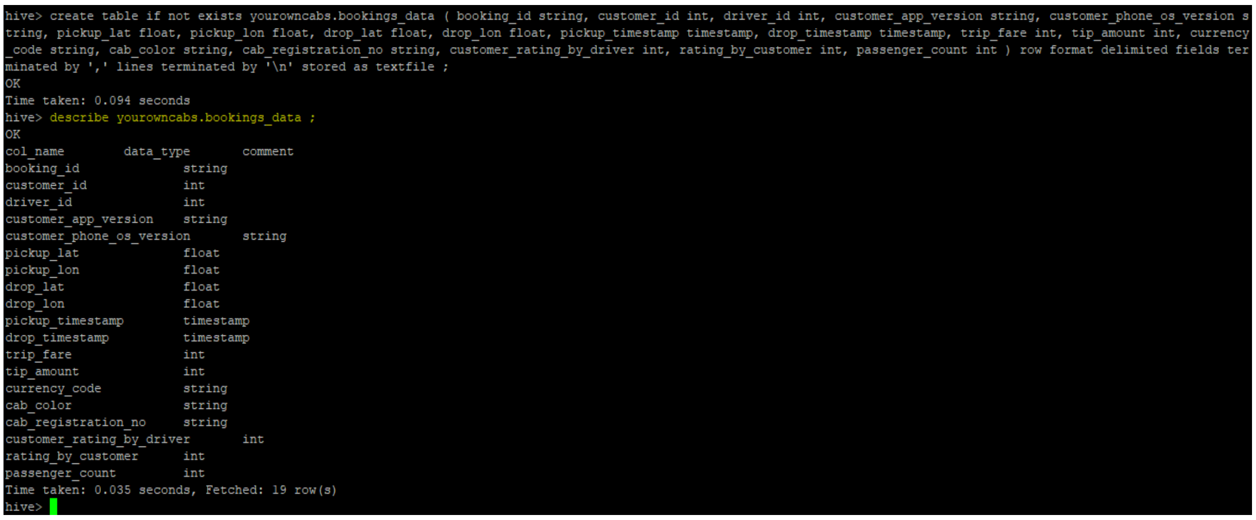
  Command:

  create table if not exists yourowncabs.bookings_data (booking_id string, customer_id int, driver_id int, customer_app_version string, customer_phone_os_version string, pickup_lat float, pickup_lon float, drop_lat float, drop_lon float, pickup_timestamp timestamp, drop_timestamp timestamp, trip_fare int, tip_amount int, currency_code string, cab_color string, cab_registration_no string, customer_rating_by_driver int, rating_by_customer int, passenger_count int) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile ;

- Below is the command used to describe the bookings_data table.

  Command:
  Describe yourowncabs.bookings_data ;

- Below is the screenshot of the bookings_data table created.



```
hive> create table if not exists yourowncabs.bookings_data ( booking_id string, customer_id int, driver_id int, customer_app_version string, customer_phone_os_version s
tring, pickup_lat float, pickup_lon float, drop_lat float, drop_lon float, pickup_timestamp timestamp, drop_timestamp timestamp, trip_fare int, tip_amount int, currency
_code string, cab_color string, cab_registration_no string, customer_rating_by_driver int, rating_by_customer int, passenger_count int ) row format delimited fields ter
minated by ',' lines terminated by '\n' stored as textfile ;
OK
Time taken: 0.094 seconds
hive> describe yourowncabs.bookings_data ;
OK
col_name        data_type       comment
booking_id              string
customer_id             int
driver_id               int
customer_app_version    string
customer_phone_os_version       string
pickup_lat              float
pickup_lon              float
drop_lat                float
drop_lon                float
pickup_timestamp        timestamp
drop_timestamp          timestamp
trip_fare               int
tip_amount              int
currency_code           string
cab_color               string
cab_registration_no     string
customer_rating_by_driver       int
rating_by_customer      int
passenger_count         int
Time taken: 0.035 seconds, Fetched: 19 row(s)
hive>
```

- Command used to create hive managed table for **date-wise total bookings**.

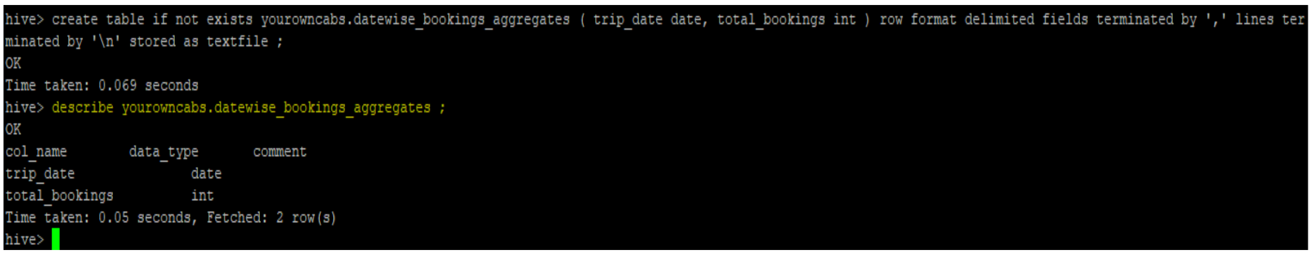
  Command:

  create table if not exists yourowncabs.datewise_bookings_aggregates (trip_date date, total_bookings int) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile ;

- Below is the command used to describe the datewise_bookings_aggregates table.

  Command:
  Describe yourowncabs.datewise_bookings_aggregates ;

- Below is the screenshot of datewise_bookings_aggregates table created.



```
hive> create table if not exists yourowncabs.datewise_bookings_aggregates ( trip_date date, total_bookings int ) row format delimited fields terminated by ',' lines ter
minated by '\n' stored as textfile ;
OK
Time taken: 0.069 seconds
hive> describe yourowncabs.datewise_bookings_aggregates ;
OK
col_name        data_type       comment
trip_date               date
total_bookings          int
Time taken: 0.05 seconds, Fetched: 2 row(s)
hive>
```

Below are commands used to load the data into the Hive tables.

- Command used to load the data in the **clickstream_data** table.

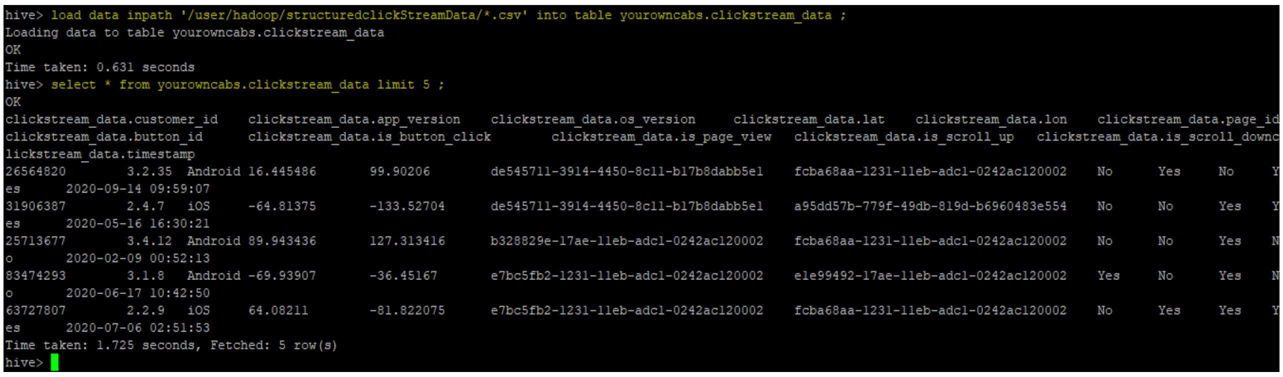
  Command:

  load data inpath '/user/hadoop/structuredclickStreamData/*.csv' into table yourowncabs.clickstream_data ;

- Below is the command used to list the first 5 rows of the clickstream_data table to verify the data.

  Command:
  select * from yourowncabs.clickstream_data limit 5 ;

- Below is the screenshot of the data loaded in the clickstream_data table.



```
hive> load data inpath '/user/hadoop/structuredclickStreamData/*.csv' into table yourowncabs.clickstream_data ;
Loading data to table yourowncabs.clickstream_data
OK
Time taken: 0.631 seconds
hive> select * from yourowncabs.clickstream_data limit 5 ;
OK
clickstream_data.customer_id    clickstream_data.app_version    clickstream_data.os_version     clickstream_data.lat    clickstream_data.lon    clickstream_data.page_id
clickstream_data.button_id      clickstream_data.is_button_click        clickstream_data.is_page_view   clickstream_data.is_scroll_up   clickstream_data.is_scroll_downc
lickstream_data.timestamp
26564820        3.2.35  Android 16.445486       99.90206        de545711-3914-4450-8c11-b17b8dabb5e1    fcba68aa-1231-11eb-adc1-0242ac120002    No      Yes     No      Y
es      2020-09-14 09:59:07
31906387        2.4.7   iOS     -64.81375       -133.52704      de545711-3914-4450-8c11-b17b8dabb5e1    a95dd57b-779f-49db-819d-b6960483e554    No      No      Yes     Y
es      2020-05-16 16:30:21
25713677        3.4.12  Android 89.943436       127.313416      b328829e-17ae-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    No      No      Yes     N
o       2020-02-09 00:52:13
83474293        3.1.8   Android -69.93907       -36.45167       e7bc5fb2-1231-11eb-adc1-0242ac120002    e1e99492-17ae-11eb-adc1-0242ac120002    Yes     No      Yes     N
o       2020-06-17 10:42:50
63727807        2.2.9   iOS     64.08211        -81.822075      e7bc5fb2-1231-11eb-adc1-0242ac120002    fcba68aa-1231-11eb-adc1-0242ac120002    No      Yes     Yes     Y
es      2020-07-06 02:51:53
Time taken: 1.725 seconds, Fetched: 5 row(s)
hive>
```

- Command used to load the data in **bookings_data** table.

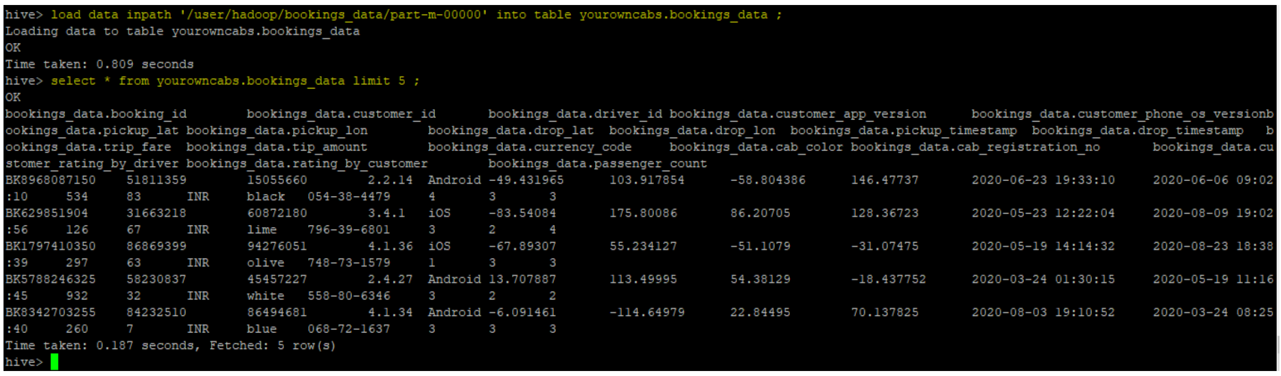
  Command:

  load data inpath '/user/hadoop/bookings_data/part-m-00000' into table yourowncabs.bookings_data ;

- Below is the command used to list the first 5 rows of the bookings_data table to verify the data.

  Command:
  select * from yourowncabs.bookings_data limit 5 ;

- Below is the screenshot of the data loaded in the bookings_data table.



```
hive> load data inpath '/user/hadoop/bookings_data/part-m-00000' into table yourowncabs.bookings_data ;
Loading data to table yourowncabs.bookings_data
OK
Time taken: 0.809 seconds
hive> select * from yourowncabs.bookings_data limit 5 ;
OK
bookings_data.booking_id        bookings_data.customer_id       bookings_data.driver_id bookings_data.customer_app_version      bookings_data.customer_phone_os_versionb
ookings_data.pickup_lat bookings_data.pickup_lon        bookings_data.drop_lat  bookings_data.drop_lon  bookings_data.pickup_timestamp  bookings_data.drop_timestamp   b
ookings_data.trip_fare  bookings_data.tip_amount        bookings_data.currency_code     bookings_data.cab_color bookings_data.cab_registration_no       bookings_data.cu
stomer_rating_by_driver bookings_data.rating_by_customer        bookings_data.passenger_count
BK8968087150    51811359        15055660        2.2.14  Android -49.431965      103.917854      -58.804386      146.47737       2020-06-23 19:33:10     2020-06-06 09:02
:10     534     83      INR     black   054-38-4479     4       3       3
BK629851904     31663218        60872180        3.4.1   iOS     -83.54084       175.80086       86.20705        128.36723       2020-05-23 12:22:04     2020-08-09 19:02
:56     126     67      INR     lime    796-39-6801     3       2       4
BK1797410350    86869399        94276051        4.1.36  iOS     -67.89307       55.234127       -51.1079        -31.07475       2020-05-19 14:14:32     2020-08-23 18:38
:39     297     63      INR     olive   748-73-1579     1       3       3
BK5788246325    58230837        45457227        2.4.27  Android 13.707887       113.49995       54.38129        -18.437752      2020-03-24 01:30:15     2020-05-19 11:16
:45     932     32      INR     white   558-80-6346     3       2       2
BK8342703255    84232510        86494681        4.1.34  Android -6.091461       -114.64979      22.84495        70.137825       2020-08-03 19:10:52     2020-03-24 08:25
:40     260     7       INR     blue    068-72-1637     3       3       3
Time taken: 0.187 seconds, Fetched: 5 row(s)
hive>
```

- Command used to load the data in **datewise_bookings_aggregates** table.

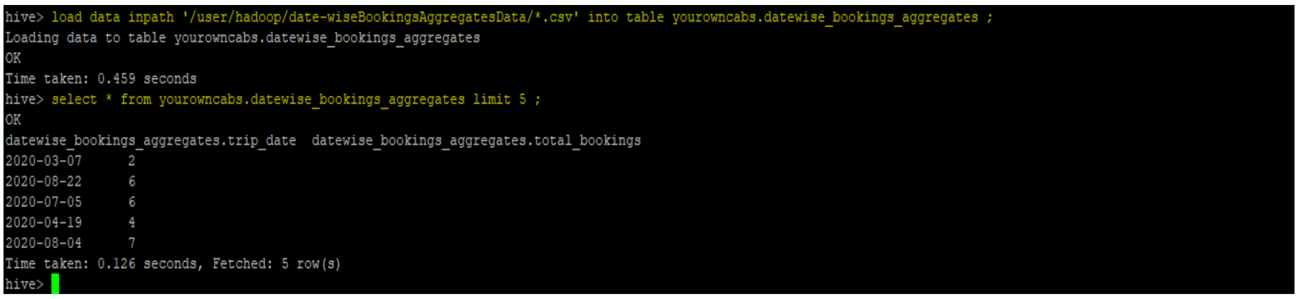
  Command:

  load data inpath '/user/hadoop/date-wiseBookingsAggregatesData/*.csv' into table yourowncabs.datewise_bookings_aggregates ;

- Below is the command used to list the first 5 rows of the datewise_bookings_aggregates table to verify the data.

  Command:
  select * from yourowncabs.datewise_bookings_aggregates limit 5 ;

- Below is the screenshot of the data loaded in the datewise_bookings_aggregates.



```
hive> load data inpath '/user/hadoop/date-wiseBookingsAggregatesData/*.csv' into table yourowncabs.datewise_bookings_aggregates ;
Loading data to table yourowncabs.datewise_bookings_aggregates
OK
Time taken: 0.459 seconds
hive> select * from yourowncabs.datewise_bookings_aggregates limit 5 ;
OK
datewise_bookings_aggregates.trip_date  datewise_bookings_aggregates.total_bookings
2020-03-07      2
2020-08-22      6
2020-07-05      6
2020-04-19      4
2020-08-04      7
Time taken: 0.126 seconds, Fetched: 5 row(s)
hive>
```