## 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.

**Task1** : **Write a job to consume clickstream data from Kafka and ingest to Hadoop.**

**Steps**:

1. Import all libraries required for running the code.

import os

import sys

os.environ["PYSPARK\_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"

os.environ["JAVA\_HOME"] = "/usr/java/jdk1.8.0\_161/jre"

os.environ["SPARK\_HOME"] = "/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"

os.environ["PYLIB"] = os.environ["SPARK\_HOME"] + "/python/lib"

sys.path.insert(0, os.environ["PYLIB"] + "/py4j-0.10.6-src.zip")

sys.path.insert(0, os.environ["PYLIB"] + "/pyspark.zip")

from pyspark.sql import SparkSession

from pyspark.sql.functions import \*

1. Initialize the spark session and set app name as Kafka to local

spark = SparkSession \

.builder \

.appName("Kafka-to-local") \

.getOrCreate()

1. Read data from Kafka by subscribing to **de-capstone3 topic**

df = spark.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "18.211.252.152:9092") \

.option("startingOffsets", "earliest") \

.option("subscribe", "de-capstone3") \

.load()

1. Drop the columns which are not required and change the column name of value to value\_str

df= df \

.withColumn('value\_str',df['value'].cast('string').alias('key\_str')).drop('value') \

.drop('key','topic','partition','offset','timestamp','timestampType')

1. Write the data to hadoop using below code. We have to use path where the raw clickstream data is stored as well as checkpoint path in HDFS

df.writeStream \

.format("json") \

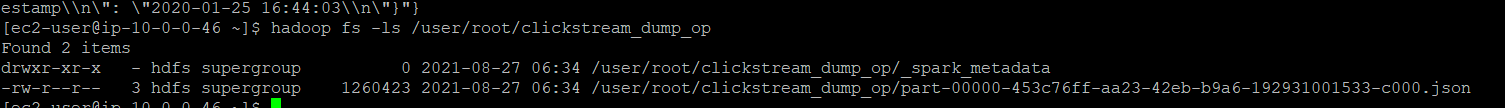
.outputMode("append") \

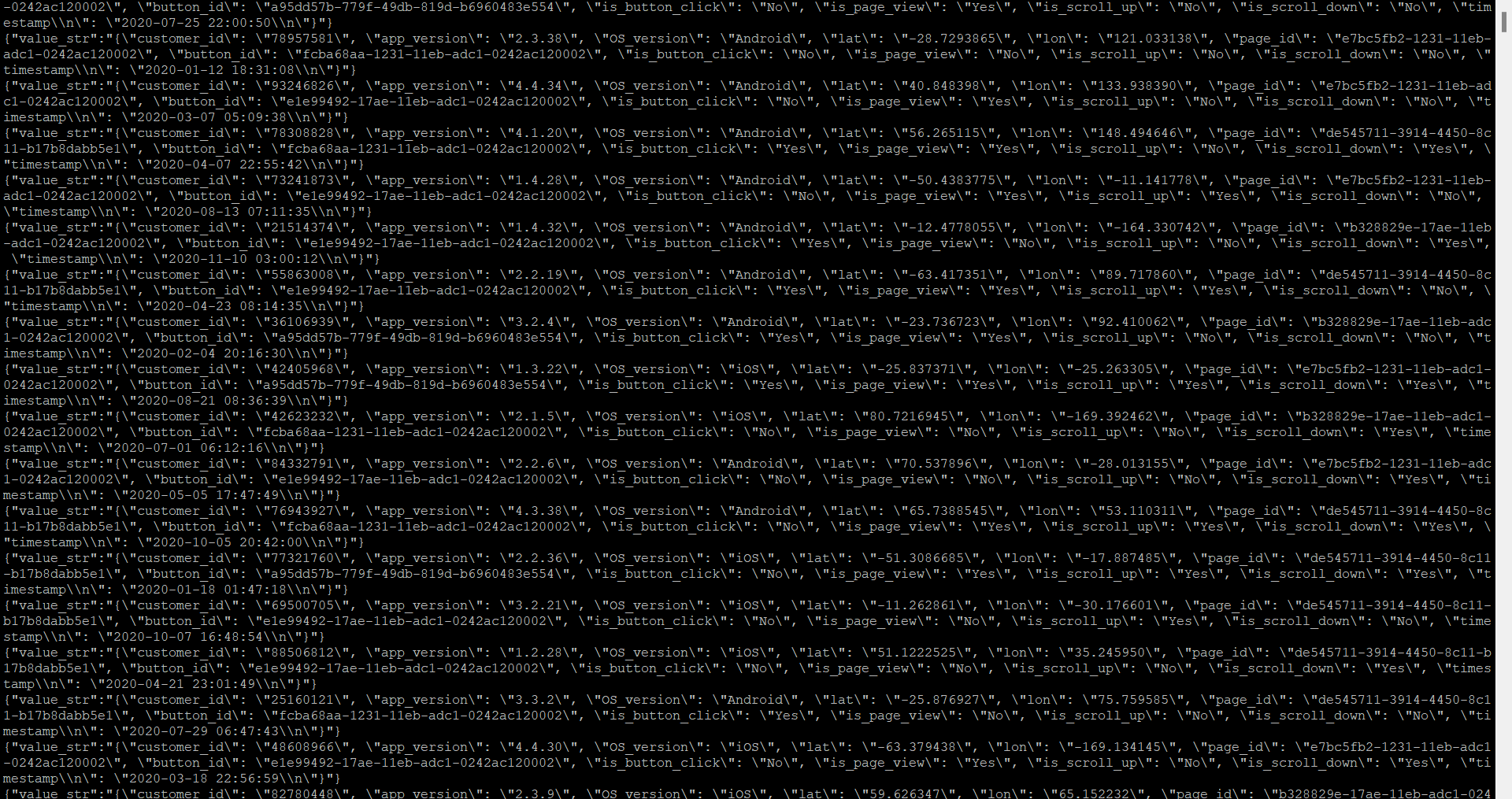
.option("path", "/user/root/clickstream\_dump\_op") \

.option("checkpointLocation", "/user/root/clickstream\_dump\_cp") \

.start() \

.awaitTermination()

****



After loading data from Kafka to hadoop, we must flatten the data for analysis. So, I have created spark\_local\_flatten.py using pyspark.

**Task 2: Write a script to ingest the relevant bookings data from AWS RDS to Hadoop.**

1. Import all libraries required for running the code.

import os

import sys

os.environ["PYSPARK\_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"

os.environ["JAVA\_HOME"] = "/usr/java/jdk1.8.0\_161/jre"

os.environ["SPARK\_HOME"] = "/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"

os.environ["PYLIB"] = os.environ["SPARK\_HOME"] + "/python/lib"

sys.path.insert(0, os.environ["PYLIB"] + "/py4j-0.10.6-src.zip")

sys.path.insert(0, os.environ["PYLIB"] + "/pyspark.zip")

from pyspark.sql import SparkSession

from pyspark.sql.functions import \*

1. Initialize the Spark Session and name app Name as Kafka to HDFS

spark=SparkSession.builder.appName("Kafka-to-HDFS").master("local").getOrCreate()

spark

1. Read raw json file which is stored in HDFS.

df=spark.read.json("clickstream.json")

1. Select the columns for analysis.

df=df.select(get\_json\_object(df['value\_str'],"$.customer\_id").alias("customer\_id"),

get\_json\_object(df['value\_str'],"$.app\_version").alias("app\_version"),

get\_json\_object(df['value\_str'],"$.OS\_version").alias("OS\_version"),

get\_json\_object(df['value\_str'],"$.lat").alias("lat"),

get\_json\_object(df['value\_str'],"$.lon").alias("lon"),

get\_json\_object(df['value\_str'],"$.page\_id").alias("page\_id"),

get\_json\_object(df['value\_str'],"$.button\_id").alias("button\_id"),

get\_json\_object(df['value\_str'],"$.is\_button\_click").alias("is\_button\_click"),

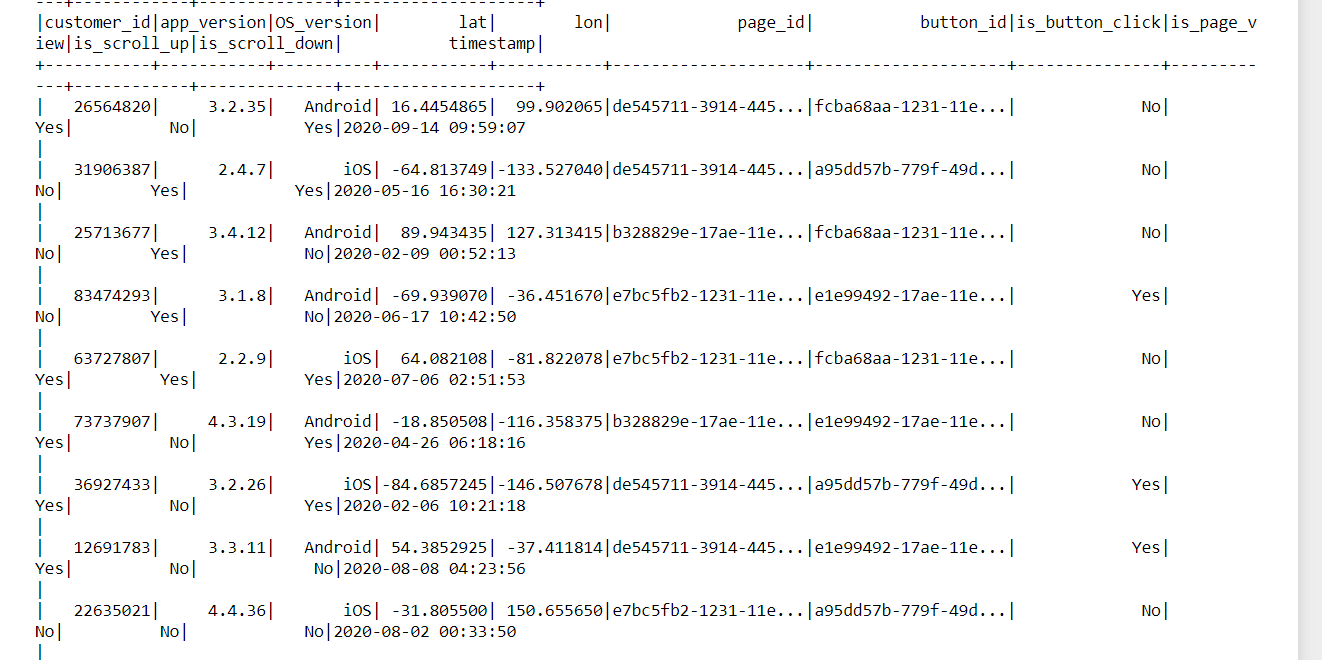
get\_json\_object(df['value\_str'],"$.is\_page\_view").alias("is\_page\_view"),

get\_json\_object(df['value\_str'],"$.is\_scroll\_up").alias("is\_scroll\_up"),

get\_json\_object(df['value\_str'],"$.is\_scroll\_down").alias("is\_scroll\_down"),

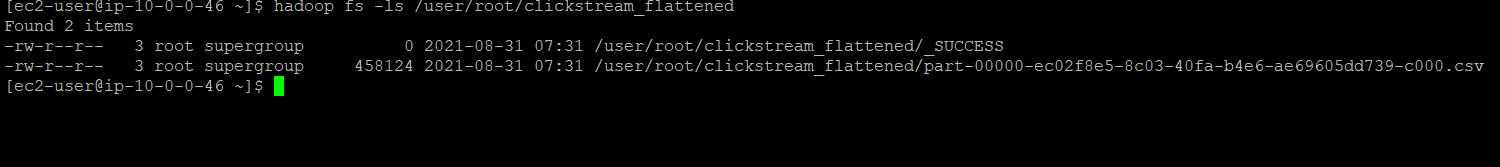
get\_json\_object(df['value\_str'],"$.timestamp").alias("timestamp")

)



1. Write the flattened data to HDFS.

df.coalesce(1).write.format('csv').mode('overwrite').save('/user/root/clickstream\_flattened',header='true')



**Task 3: Create aggregates for finding date-wise total bookings using the Spark script.**

Before performing aggregates for datewise bookings, we have to load the data from AWS RDS to Hadoop using Sqoop import job. Below is the sqoop import job.

sqoop import \

--connect jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase \

--table bookings \

--username student --password STUDENT123 \

--target-dir /user/root/cab\_rides \

-m 1

1. First import all required libraries

import os

import sys

os.environ["PYSPARK\_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"

os.environ["JAVA\_HOME"] = "/usr/java/jdk1.8.0\_232-cloudera/jre"

os.environ["SPARK\_HOME"]="/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"

os.environ["PYLIB"] = os.environ["SPARK\_HOME"] + "/python/lib"

sys.path.insert(0, os.environ["PYLIB"] +"/py4j-0.10.6-src.zip")

sys.path.insert(0, os.environ["PYLIB"] +"/pyspark.zip")

from pyspark.sql import SparkSession

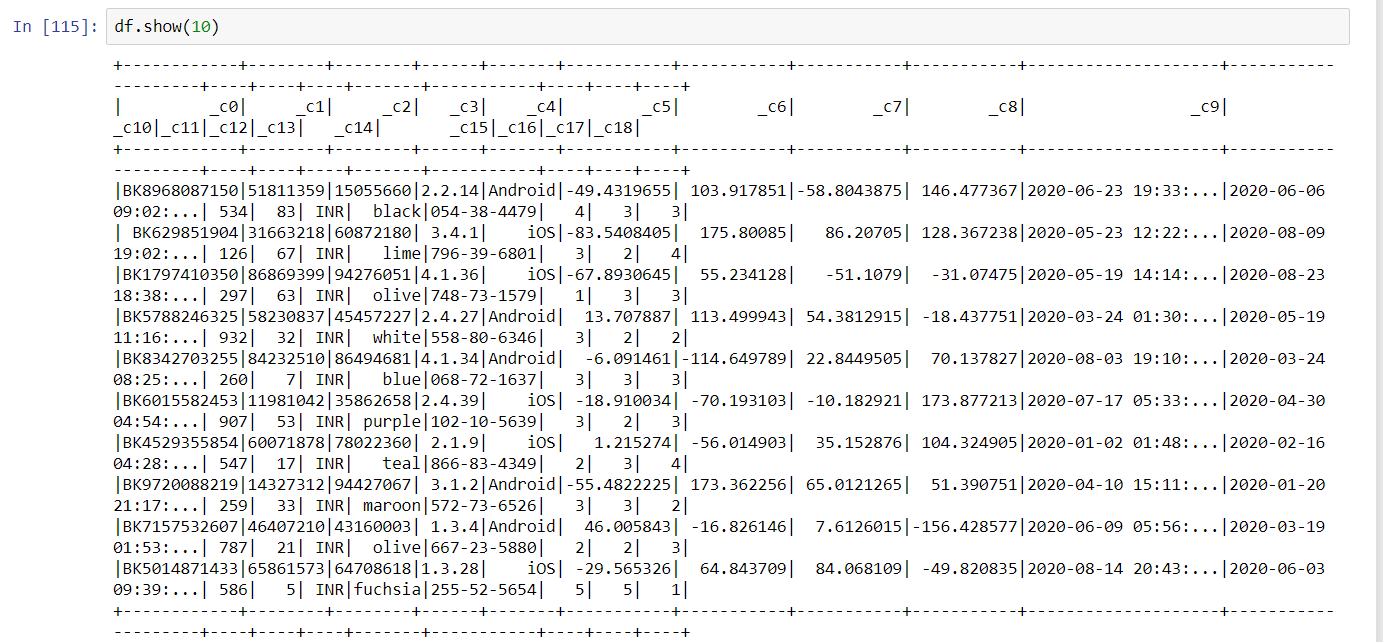
from pyspark.sql.functions import \*

1. Initialize the Spark session.

spark=SparkSession.builder.appName("datewise\_bookings\_aggregates\_spark").master("local").getOrCreate()

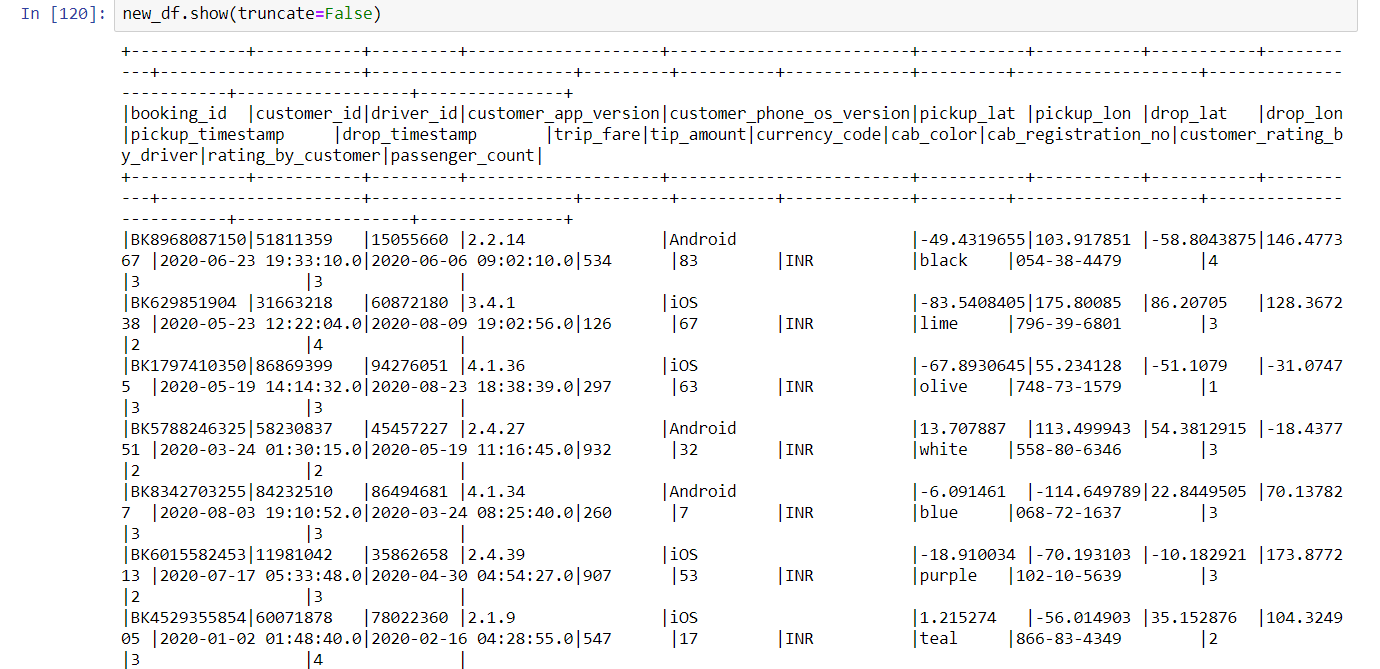
1. Read the data from HDFS which was loaded using Sqoop.

df=spark.read.csv("/user/root/cab\_rides/part-m-00000")



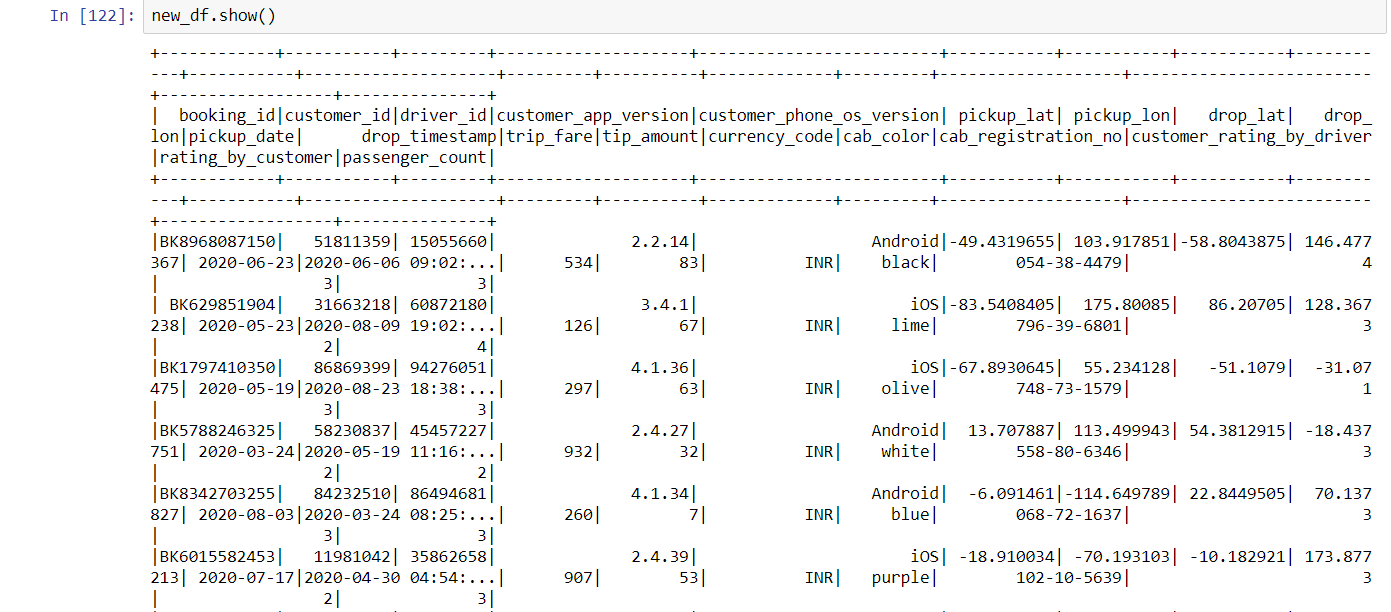
1. Rename the columns

* new\_col = ["booking\_id","customer\_id","driver\_id","customer\_app\_version","customer\_phone\_os\_version","pickup\_lat","pickup\_lon","drop\_lat","drop\_lon","pickup\_timestamp","drop\_timestamp","trip\_fare","tip\_amount","currency\_code","cab\_color","cab\_registration\_no","customer\_rating\_by\_driver","rating\_by\_customer","passenger\_count"]
* new\_df = df.toDF(\*new\_col)



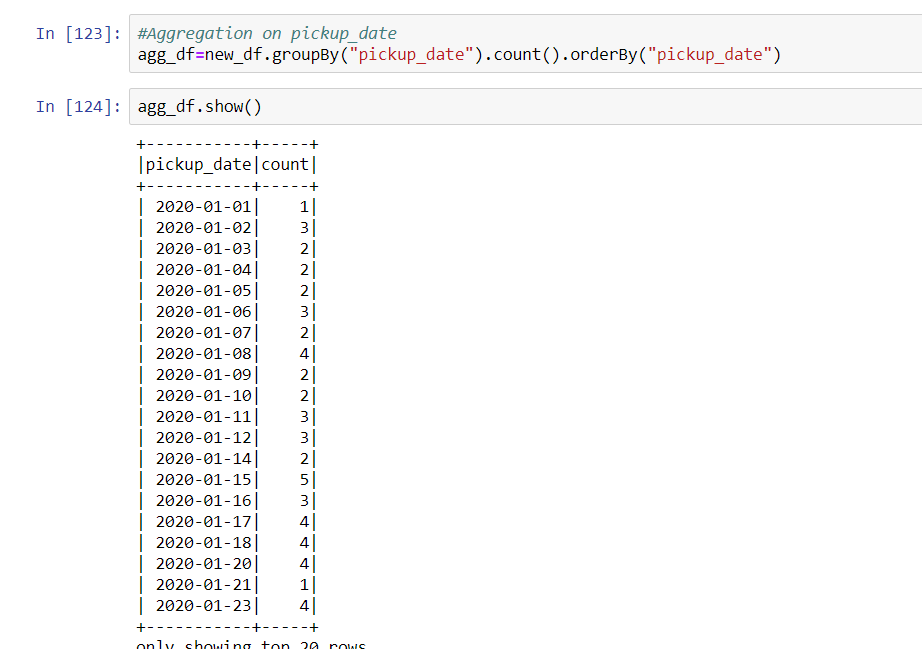
1. Converting pickup\_timestamp to date by extracting date from pickup\_timestamp for aggregation

new\_df=new\_df.select("booking\_id","customer\_id","driver\_id","customer\_app\_version","customer\_phone\_os\_version","pickup\_lat","pickup\_lon","drop\_lat", "drop\_lon",to\_date(col('pickup\_timestamp')).alias('pickup\_date').cast("date"),"drop\_timestamp","trip\_fare","tip\_amount","currency\_code","cab\_color","cab\_registration\_no","customer\_rating\_by\_driver", "rating\_by\_customer","passenger\_count")



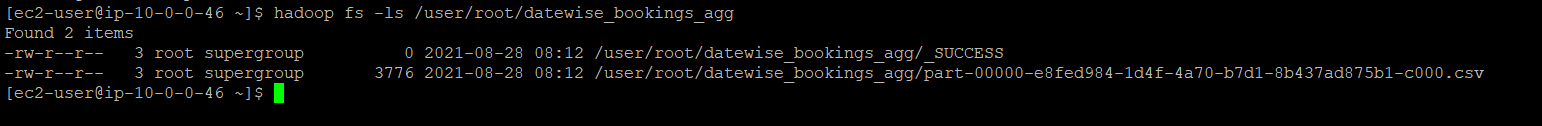
1. Aggregation on pickup date.

agg\_df=new\_df.groupBy("pickup\_date").count().orderBy("pickup\_date")



1. Command to move above grouped data into HDFS.

agg\_df.coalesce(1).write.format('csv').mode('overwrite').save('/user/root/datewise\_bookings\_agg',header='true')



**Task 4:**

* **Create a Hive-managed table for clickstream data.**
* **Create a Hive-managed table for bookings data.**
* **Create a Hive-managed table for aggregated data in Task 3.**

For Creating tables in Hive, first we must create databases. I have created capstone database in Hive. The tables which I have created for all three tasks are clickstream, Booking, Aggregate\_datewise. Here the data is stored in csv format, so while creating tables, we have to use fields terminated by , .

1. **Clickstream data: Hive Managed table.**

create table if not exists clickstream (

customer\_id int,

app\_version string,

os\_version string,

lat double,

lon double,

page\_id varchar(100),

button\_id varchar(100),

is\_button\_click string,

is\_page\_view string,

is\_scroll\_up string,

is\_scroll\_down string,

timestamp timestamp )

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

stored as textfile;

Load data into the table using below command.

load data local inpath '/root/clickstream/clickstream\_flattened.csv' into table clickstream

1. **Bookings data:**

create table if not exists booking (

booking\_id string,

customer\_id int,

driver\_id int,

customer\_app\_version string,

customer\_phone\_os\_version string,

pickup\_lat double,

pickup\_lon double,

drop\_lat double,

drop\_lon double,

pickup\_timestamp timestamp,

drop\_timestamp timestamp,

trip\_fare int,

tip\_amount int,

currency\_code string,

cab\_color string,

cab\_registration\_no int,

customer\_rating\_by\_driver varchar(100),

rating\_by\_customer int,

passenger\_count int)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

stored as textfile;

Load data into the table using below command.

load data local inpath '/root/booking\_data/part-m-00000' into table booking;

1. **Aggregated data datewise:**

create table if not exists aggregate\_datewise(

pickup\_date date,

booking\_id\_count int

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

stored as textfile;

Load data into the table using below command.

load data local inpath '/root/agg\_datewise' into table aggregate\_datewise;