Logic For First Submission

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

The whole process is distributed to be conducted in task based manner. Below describes tasks that are to be executed sequentially.

Task 1

The first steps is to get the clickstream raw data from Kafka server (provided in project resources) to Hadoop. For that script spark_kafka_to_local is written that reads from Kafka server and writes to folders on hadoop

1. We start with first importing the modules

*C:\Users\rancy\Downloads\CabRides_Kafka_to_Hadoop.py - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

new 2 | CabRides_Kafka_to_Hadoop.py

```
# Script is to get all the cab rides related click stream data residing in given Kafka server to Hadoop cluster


# we start with first importing the modules
import os,sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
```

2. Set required environment variables needed to get the data

```
# Next step is to set required environment variables needed to get the data
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")
```

3. Next initialize Spark session

```
# We will now initialize Spark session
spark = SparkSession \
        .builder \
        .appName("Kafka-to-local") \
        .getOrCreate()
```

4. Read Data from kafka server from given Kafka server details

```
# From the given server connection details connect to kafka and get the stream in a dataframe
  # Read Input from kafka
streamdf = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
        .option("startingOffsets", "earliest") \
        .option("subscribe", "de-capstone3") \
        .load()
```

5. Keep relevant field ‘value’ rename it to ‘value_str’ and drop other irrelevant fields

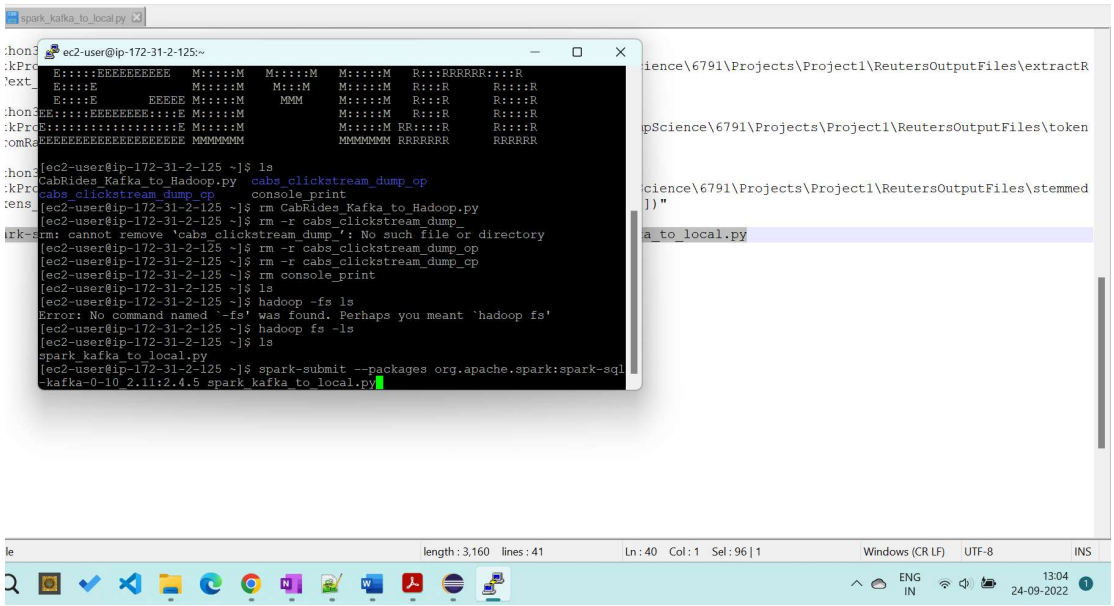
```
# get only relevant fields and drop others
streamdf= streamdf \
      .withColumn('value_str',streamdf['value'].cast('string').alias('key_str')).drop('value') \
      .drop('key','topic','partition','offset','timestamp','timestampType')
```

<Steps to load the data into Hadoop>

1. Write the click stream to folder ‘cabs_clickstream_dump_op’ in Hadoop

```
#Writing the click stream to a folder in Hadoop
streamdf.writeStream \
   .format("json") \
   .outputMode("append") \
   .option("path", "cabs_clickstream_dump_op") \
   .option("checkpointLocation", "cabs_clickstream_dump_cp") \
   .start() \
   .awaitTermination()
```

2. Submit Spark Job



Once we get the raw data we pull the raw json file and rename it to use it for flattening our raw data for better data analysis. For which we use another script named spark_local_flatten.py that has following steps:

1. Import libraries and set environment variables

```
sers\rancy\OneDrive\RancyStudiesBackup\ExecutivePGonDataScience\DataEngineering\CapStone Project\Rancy_CabRides_First_Submission\spark_local_flatten.py - Notepad++
dit  Search  View  Encoding  Language  Settings  Tools  Macro  Run  Plugins  Window  ?
spark_kafka_to_local.py   spark_local_flatten.py

# import required modules
import os
import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# set environment variables
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")

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

2. Create Spark session and dataframe that reads from raw clickstream json file

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

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

3. Select relevant fields and create aliases for data 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")
             )
```

4. Write it in CSV form to hadoop

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

Task 2

Next task is to get cab rides data from given RDS to Hadoop, for which we use following command

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

Task 3

Task 3 is to create date wise aggregate bookings and store that in a CSV file, this we do with a pyspark script and then we move this CSV to hadoop

Step 1 – import required modules and set environment variables

datewise_bookings_aggregates_spark Last Checkpoint: 3 minutes ago (autosaved)

View Insert Cell Kernel Widgets Help Trusted

Run Code nbdiff

```
import os
import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1664045986960_0003	pyspark	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```

```
# create environment variables
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")
```

Step 2 Create spark session and read raw cab rides data

```
]: # Create spark session
spark=SparkSession.builder.appName("datewise_bookings_aggregates_spark").master("local").getOrCreate()
spark
```

```
<pyspark.sql.session.SparkSession object at 0x7f1e9a231b10>
```

```
]: df=spark.read.csv("/user/root/cab_rides/part-m-00000")
```

Spark Job Progress

```
]: # Check count of data
df.count()
```

Spark Job Progress

```
1000
```

Step 3 Check count of rows, schema and first 10 rows

```
# Check count of data
df.count()
```

Spark Job Progress

```
1000
```

```
# check first 10 rows
df.show(10)
```

Spark Job Progress

```
+------------+--------+--------+------+------+-----------+-----------+-----------+-----------+--------------------+-----------
---------+----+----+----+-------+-----------+----+----+----+
|         _c0|     _c1|     _c2|   _c3|    _c4|        _c5|        _c6|        _c7|        _c8|                _c9|
_c10|_c11|_c12|_c13|   _c14|       _c15|_c16|_c17|_c18|
+------------+--------+--------+------+------+-----------+-----------+-----------+-----------+--------------------+-----------
---------+----+----+----+-------+-----------+----+----+----+
|BK8968087150|51811359|15055660|2.2.14|Android|-49.4319655| 103.917851|-58.8043875| 146.477367|2020-06-23 19:33:...|2020-06-06
09:02:...| 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:...|2020-08-09
19:02:...| 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:...|2020-08-23
18:38:...| 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:...|2020-05-19
11:16:...| 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:...|2020-03-24
08:25:...| 260|   7| INR|   blue|068-72-1637|   3|   3|   3|
```

```
# Check schema
df.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c14: string (nullable = true)
 |-- _c15: string (nullable = true)
 |-- _c16: string (nullable = true)
```

Step 4 rename columns and create new data frame

```
# Rename columns for better understanding and create new dataframe with these columns

new_col = ["booking_id","customer_id","driver_id","customer_app_version","customer_phone_os_version","pickup_lat","pickup_lon","c

new_df = df.toDF(*new_col)
```

Step 5 check new dataframe value

```
]: #validate new dataframe
   new_df.show(truncate=False)
```

Spark Job Progress

```
+------------+-----------+---------+--------------------+------------------------+-----------+-----------+-----------+--------
---+--------------------+--------------------+---------+----------+-------------+---------+-------------------+--------------
-----------+-----------------+---------------+
|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_b
y_driver|rating_by_customer|passenger_count|
+------------+-----------+---------+--------------------+------------------------+-----------+-----------+-----------+--------
---+--------------------+--------------------+---------+----------+-------------+---------+-------------------+--------------
-----------+-----------------+---------------+
|BK8968087150|51811359   |15055660 |2.2.14              |Android                 |-49.4319655|103.917851 |-58.8043875|146.4773
67 |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   |128.3672
38 |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   |-31.0747
```

Step 6 get only the dates from the date fields which have timestamps as well.

```
]: #Now convert the 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","pick
```

```
new_df.show()
```

Spark Job Progress

```
+------------+-----------+---------+--------------------+------------------------+-----------+-----------+-----------+--------
---+-----------+--------------------+---------+----------+-------------+---------+-------------------+------------------------
+------------------+---------------+
|  booking_id|customer_id|driver_id|customer_app_version|customer_phone_os_version| pickup_lat| pickup_lon|   drop_lat|   drop_
lon|pickup_date|      drop_timestamp|trip_fare|tip_amount|currency_code|cab_color|cab_registration_no|customer_rating_by_driver
|rating_by_customer|passenger_count|
+------------+-----------+---------+--------------------+------------------------+-----------+-----------+-----------+--------
---+-----------+--------------------+---------+----------+-------------+---------+-------------------+------------------------
+------------------+---------------+
|BK8968087150|   51811359| 15055660|              2.2.14|                 Android|-49.4319655| 103.917851|-58.8043875| 146.477
367| 2020-06-23|2020-06-06 09:02:...|      534|        83|          INR|    black|        054-38-4479|                      4
|                  3|              3|
| BK629851904|   31663218| 60872180|               3.4.1|                     iOS|-83.5408405|  175.80085|   86.20705| 128.367
```

Step7 create aggregate data frame by aggregating on pickup_date field

```
# create aggregate on pickup_date field
agg_df=new_df.groupBy("pickup_date").count().orderBy("pickup_date")
```

```
agg_df.show(5)
```

Spark Job Progress

```
+-----------+-----+
|pickup_date|count|
+-----------+-----+
| 2020-01-01|    1|
| 2020-01-02|    3|
| 2020-01-03|    2|
| 2020-01-04|    2|
| 2020-01-05|    2|
+-----------+-----+
only showing top 5 rows
```

Step 8 Write this aggregate dataframe as CSV to hadoop

```
#Write the aggregate csv to hadoop
agg_df.coalesce(1).write.format('csv').mode('overwrite').save('/user/root/datewise_bookings_agg',header='true')
```

Spark Job Progress

Progress:

Step 8 Verify the CSV is stored in hadoop

```
[hadoop@ip-172-31-62-179 ~]$ hadoop fs -ls /user/root
Found 3 items
drwxr-xr-x   - hadoop hadoop          0 2022-09-24 19:16 /user/root/cab_rides
drwxr-xr-x   - hadoop hadoop          0 2022-09-24 19:33 /user/root/clickstream_
flattened
drwxr-xr-x   - livy   hadoop          0 2022-09-24 20:16 /user/root/datewise_boo
kings_agg
```

Task 4

Next step is to create database and tables within it to load respective data to tables

```
hive> create database cabrides_db;
OK
Time taken: 0.398 seconds
hive> show databases;
OK
cabrides_db
default
Time taken: 0.036 seconds, Fetched: 2 row(s)
```

Create table and load clickstream data

```
hive>
    > 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;
OK
Time taken: 0.711 seconds
```

Load clickstream data to this table

```
hive> load data inpath "/user/root/clickstream_flattened/part-00000-aa5dcf04-898
a-4189-b284-5f57b3789b9b-c000.csv" into table clickstream;
Loading data to table cabrides_db.clickstream
OK
Time taken: 1.232 seconds
hive>
```

Create table and load Booking data

```
hive> 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;
OK
Time taken: 0.094 seconds
```

Load booking data to this table

```
hive> load data inpath "/user/root/cab_rides/part-m-00000" into table booking;
Loading data to table cabrides_db.booking
OK
Time taken: 0.581 seconds
```

Create table and load aggregate date data

```
hive> create table if not exists aggregate_datewise(
    >         pickup_date date,
    >         booking_id_count int
    >         )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > stored as textfile;
OK
Time taken: 0.088 seconds
```

Load aggregate_datewise data to this table

```
hive> load data inpath "/user/root/datewise_bookings_agg" into table aggregate_d
atewise;
Loading data to table cabrides_db.aggregate_datewise
OK
Time taken: 0.543 seconds
```

Check tables

```
hive> show tables;
OK
aggregate_datewise
booking
clickstream
Time taken: 0.056 seconds, Fetched: 3 row(s)
```

```
hive> SELECT COUNT(*) FROM clickstream;
Query ID = hadoop_20220924210528_9f64e343-5791-4d71-8a63-6460252ad1ba
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1664045986960
_0005)

Map 1: 0/1      Reducer 2: 0/1
Map 1: 0/1      Reducer 2: 0/1
Map 1: 0(+1)/1  Reducer 2: 0/1
Map 1: 0/1      Reducer 2: 0/1
Map 1: 1/1      Reducer 2: 0(+1)/1
Map 1: 1/1      Reducer 2: 0/1
Map 1: 1/1      Reducer 2: 1/1
OK
3001
```

```
hive> SELECT COUNT(*) FROM booking;
Query ID = hadoop_20220924210555_8dbd6f73-12a9-4a11-ac31-0f27ecc93aef
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1664045986960
_0005)

Map 1: 0/1      Reducer 2: 0/1
Map 1: 0/1      Reducer 2: 0/1
Map 1: 0(+1)/1  Reducer 2: 0/1
Map 1: 1/1      Reducer 2: 0/1
Map 1: 1/1      Reducer 2: 0(+1)/1
Map 1: 1/1      Reducer 2: 1/1
OK
1000
Time taken: 6.088 seconds, Fetched: 1 row(s)
```

```
hive> SELECT COUNT(*) FROM aggregate_datewise;
Query ID = hadoop_20220924210730_cd44cec0-1d6f-4406-b61f-857cb6fb52c3
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1664045986960
_0005)

Map 1: 0/1      Reducer 2: 0/1
Map 1: 0(+1)/1  Reducer 2: 0/1
Map 1: 1/1      Reducer 2: 0(+1)/1
Map 1: 1/1      Reducer 2: 1/1
OK
290
Time taken: 5.322 seconds, Fetched: 1 row(s)
```