

Logic For First Submission

Step1: Writing of job to consume clickstream data from Kafka and ingest to Hadoop 1

1.1 Created a python file for the code which will ingest the relevant data from Kafka into hadoop.
vi spark_kafka_to_local.py

1.2 Explaining the code in spark_kafka_to_local.py

- Necessary imports

```
import os
import sys

//Environment setup
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")

//Selective Imports
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
```

- Creating a Spark Session for performing actions like reading of data.
spark = SparkSession \
 .builder \
 .appName("Kafka-to-local") \
 .getOrCreate()
- Reading of data from Kafka topic into a data frame to make an action ready data.
df = spark.readStream \
 .format("kafka") \
 .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
 .option("startingOffsets", "earliest") \
 .option("subscribe", "de-capstone3") \
 .load()

- Dropping irrelevant columns from the data frame and make data more relevant to the use case.

```
df= df \ 
    .withColumn('value_str',df['value'].cast('string').alias('key_str')).drop('value') \
    .drop('key','topic','partition','offset','timestamp','timestampType')
```

- Writing data from Kafka topic to HDFS for more actions to be performed, using awaitTermination to protect and keep running until terminated.

```
df.writeStream \ 
    .format("json") \ 
    .outputMode("append") \ 
    .option("path", "/user/root/clickstream_dump_op") \ 
    .option("checkpointLocation", "/user/root/clickstream_dump_cp") \ 
    .start() \ 
    .awaitTermination()
```

Step2: Executing Spark Submit command, to ingest the relevant data from Kafka into hadoop.

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark_kafka_to_local.py
18.211.252.152 9092 de-capstone3
```

Screenshot showing spark command run

```
[hadoop@ip-172-31-8-156 ~]$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark_kafka_to_local.py 18.211.252.152 9092 de-capstone3
Ivy Default Cache set to: /home/hadoop/.ivy2/cache
The jars for the packages stored in: /home/hadoop/.ivy2/jars
:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.spark#spark-sql-kafka-0-10_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-57e07081-cc45-41dc-a2de-a757c3704b0a;1.0
  confs: [default]
  found org.apache.spark#spark-sql-kafka-0-10_2.11;2.4.5 in central
  found org.apache.kafka#kafka-clients;2.0.0 in central
  found org.lz4#lz4-java;1.4.0 in central
  found org.xerial.snappy#snappy-java;1.1.7.3 in central
  found org.slf4j#slf4j-api;1.7.16 in central
  found org.spark-project.spark#unused;1.0.0 in central
downloading https://repo.maven.org/maven2/org/apache/spark/spark-sql-kafka-0-10_2.11/2.4.5/spark-sql-kafka-0-10_2.11-2.4.5.jar ...
  [SUCCESSFUL ] org.apache.spark#spark-sql-kafka-0-10_2.11-2.4.5!spark-sql-kafka-0-10_2.11.jar (32ms)
downloading https://repo.maven.org/maven2/org/apache/kafka/kafka-clients/2.0.0/kafka-clients-2.0.0.jar ...
  [SUCCESSFUL ] org.apache.kafka#kafka-clients;2.0.0!kafka-clients.jar (93ms)
downloading https://repo.maven.org/maven2/org/spark-project/spark/unused/1.0.0/unused-1.0.0.jar ...
  [SUCCESSFUL ] org.spark-project.spark#unused;1.0.0!unused.jar (4ms)
downloading https://repo.maven.org/maven2/org/lz4/lz4-java/1.4.0/lz4-java-1.4.0.jar ...
  [SUCCESSFUL ] org.lz4#lz4-java;1.4.0!lz4-java.jar (22ms)
downloading https://repo.maven.org/maven2/org/xerial/snappy/snappy-java/1.1.7.3/snappy-java-1.1.7.3.jar ...
  [SUCCESSFUL ] org.xerial.snappy#snappy-java;1.1.7.3!snappy-java.jar(bundle) (89ms)
downloading https://repo.maven.org/maven2/org/slf4j/slf4j-api/1.7.16/slf4j-api-1.7.16.jar ...
  [SUCCESSFUL ] org.slf4j#slf4j-api;1.7.16!slf4j-api.jar (4ms)
:: resolution report :: resolve 1799ms :: artifacts dl 254ms
  :: modules in use:
  org.apache.kafka#kafka-clients;2.0.0 from central in [default]
  org.apache.spark#spark-sql-kafka-0-10_2.11;2.4.5 from central in [default]
  org.lz4#lz4-java;1.4.0 from central in [default]
  org.slf4j#slf4j-api;1.7.16 from central in [default]
  org.spark-project.spark#unused;1.0.0 from central in [default]
  org.xerial.snappy#snappy-java;1.1.7.3 from central in [default]
-----
|           |           modules           ||   artifacts   |
|   conf     |   number| search|dwnlded|evicted||   number|dwnlded|
-----|   default  |   6   |   6   |   6   |   0   ||   6   |   6   |
```

Screenshot showing file data in structured format

48927284, 1, 3, 12, 108, 5, 9788155, 151, 0.64029, e7bc5fb2-1231-11eb-adcl-0242ac120002, fcbab6aa-1231-11eb-adcl-0242ac120002, No, Yes, No, No, "",
85177527, 4, 2, 10, 108, -51, 8301325, -70, 399319, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, No, Yes, "",
47743867, 3, 3, 20, 108, -13, 990733, -178, 175285, b328829e-17ae-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, No, No, "",
46694536, 4, 2, 31, 108, -59, 551650, 115, 343099, e7bc5fb2-1231-11eb-adcl-0242ac120002, fcbab6aa-1231-11eb-adcl-0242ac120002, Yes, No, Yes, Yes, "",
87861168, 4, 4, 27, 108, -9, 0_299885, 87, 753166, de545711-3914-4450-8c11-b17bdabb5e1, fcbab6aa-1231-11eb-adcl-0242ac120002, Yes, No, Yes, "",
98750961, 4, 4, 24, 108, -67, 056611, 42, 210252, b328829e-17ae-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, Yes, Yes, Yes, No, "",
34179709, 2, 4, 1, 108, -65, 870590, 112, 347735, de545711-3914-4450-8c11-b17bdabb5e1, ele99492-17ae-11eb-adcl-0242ac120002, Yes, No, Yes, Yes, "",
35577566, 2, 2, 18, 108, 48, 28, 5281265, 28, 317597, e7bc5fb2-1231-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, Yes, Yes, Yes, No, "",
39790044, 3, 3, 38, 108, 60, 0575415, -36, 785274, b328829e-17ae-11eb-adcl-0242ac120002, fcbab6aa-1231-11eb-adcl-0242ac120002, No, Yes, Yes, No, "",
45451715, 2, 3, 8, 108, 77, 0398825, 174, 942366, e7bc5fb2-1231-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, Yes, Yes, No, Yes, "",
87559592, 2, 2, 17, 108, -72, 2719865, -66, 732794, e7bc5fb2-1231-11eb-adcl-0242ac120002, fcbab6aa-1231-11eb-adcl-0242ac120002, No, Yes, No, No, "",
20011383, 2, 2, 9, 108, Android, -59, 551650, 115, 343099, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, No, No, Yes, No, "",
03519078, 4, 2, 9, 108, Android, 78, 445605, -83, 204209, b328829e-17ae-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, No, No, Yes, No, "",
35594325, 4, 3, 11, 108, 29, 422319, 106, 217372, b328829e-17ae-11eb-adcl-0242ac120002, fcbab6aa-1231-11eb-adcl-0242ac120002, No, Yes, Yes, Yes, "",
79134831, 1, 3, 28, 108, -43, 676692, -13, 693995, de545711-3914-4450-8c11-b17bdabb5e1, ele99492-17ae-11eb-adcl-0242ac120002, Yes, Yes, Yes, No, "",
74932660, 2, 4, 11, 108, Android, -16, 250308, -67, 90392, b328829e-17ae-11eb-adcl-0242ac120002, No, No, No, Yes, "",
45594816, 1, 2, 37, 108, 28, 116246, 80, 131224, b328829e-17ae-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, No, Yes, "",
70449656, 2, 3, 28, 108, -39, 571418, 108, 352612, de545711-3914-4450-8c11-b17bdabb5e1, ele99492-17ae-11eb-adcl-0242ac120002, No, Yes, No, Yes, "",
41398860, 4, 1, 10, 108, Android, -58, 4709445, -56, 966952, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, No, No, Yes, Yes, "",
35844721, 2, 3, 22, 108, -39, 571418, -56, 869833, de545711-3914-4450-8c11-b17bdabb5e1, fcbab6aa-1231-11eb-adcl-0242ac120002, Yes, Yes, No, Yes, "",
49155394, 4, 1, 38, 108, Android, -65, 173138, -43, 282215, de545711-3914-4450-8c11-b17bdabb5e1, fcbab6aa-1231-11eb-adcl-0242ac120002, No, No, No, Yes, "",
9247259, 1, 1, 28, 108, -29, 422319, 106, 976049, e7bc5fb2-1231-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, Yes, No, No, No, "",
39856207, 4, 4, 21, 108, -45, 9151095, -83, 708611, de545711-3914-4450-8c11-b17bdabb5e1, ele99492-17ae-11eb-adcl-0242ac120002, Yes, No, No, Yes, "",
47664043, 4, 3, 14, 108, Android, -11, 742446, 111, 9308952, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, No, Yes, "",
39790045, 1, 1, 32, 108, -86, 7711605, -86, 608955, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, Yes, Yes, No, "",
50208693, 1, 2, 1, 108, -25, 7465225, -115, 986705, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, No, No, Yes, No, "",
21248817, 1, 2, 1, 108, -25, 7465225, -115, 986705, e7bc5fb2-1231-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, Yes, No, Yes, No, "",
52029795, 2, 2, 30, 108, -14, 5909615, -168, 408479, b328829e-17ae-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, Yes, No, "",
78298369, 4, 1, 11, 108, -33, 7870705, -128, 420903, b328829e-17ae-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, Yes, No, "",
74700232, 3, 1, 6, 108, 28, 574094, -26, 254748, b328829e-17ae-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, No, No, No, Yes, "",
52434368, 4, 4, 13, 108, Android, -96, 2803345, -4, 684262, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, No, Yes, No, No, "",
83053436, 1, 1, 25, 108, 69, 6784285, -33, 018506, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, Yes, Yes, No, "",
36425257, 4, 3, 9, 108, 86, 945103, -171, 605410, b328829e-17ae-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, No, No, "",
89571902, 1, 1, 38, 108, -64, -64, 4794635, 9, 497102, de545711-3914-4450-8c11-b17bdabb5e1, ele99492-17ae-11eb-adcl-0242ac120002, No, No, No, Yes, "",
87051710, 4, 2, 3, 108, -5, 041723, -29, 906932, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, No, No, "",
84077369, 1, 3, 36, 108, Android, 83, 702958, -41, 236421, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, No, Yes, Yes, No, "",
30901771, 1, 1, 32, 108, -33, 601921, -18, 663633, de545711-3914-4450-8c11-b17bdabb5e1, fcbab6aa-1231-11eb-adcl-0242ac120002, No, No, Yes, Yes, "",
67047768, 4, 3, 3, 108, 50, 86, 817671, -21, 621528, de545711-3914-4450-8c11-b17bdabb5e1, fcbab6aa-1231-11eb-adcl-0242ac120002, Yes, No, Yes, Yes, "",
50945495, 4, 3, 21, 108, Android, -19, 0183955, -20, 792102, de545711-3914-4450-8c11-b17bdabb5e1, fcbab6aa-1231-11eb-adcl-0242ac120002, No, No, No, Yes, "",
67036669, 4, 1, 9, 108, Android, -68, 149568, -37, 836674, b328829e-17ae-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, No, Yes, Yes, No, "",
27315595, 4, 1, 18, 108, -177, 174549, de545711-3914-4450-8c11-b17bdabb5e1, a95dd57b-779f-49db-819d-b6960483e554, Yes, Yes, No, No, "",
70426942, 2, 3, 13, 108, 76, 747663, -93, 546816, b328829e-17ae-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, Yes, No, Yes, Yes, "",
10895363, 1, 1, 1, 108, -10, -10, 4794635, 49, 705580, de545711-3914-4450-8c11-b17bdabb5e1, ele99492-17ae-11eb-adcl-0242ac120002, Yes, No, No, No, "",
99247595, 2, 3, 15, 108, Android, 15, 9023165, 49, 306042, e7bc5fb2-1231-11eb-adcl-0242ac120002, a95dd57b-779f-49db-819d-b6960483e554, No, Yes, No, No, "",
63762620, 4, 1, 40, 108, 60, 507770, 56, 980517, de545711-3914-4450-8c11-b17bdabb5e1, ele99492-17ae-11eb-adcl-0242ac120002, No, No, Yes, No, "",
84418220, 4, 1, 40, 108, 64, 64043925, -53, 285161, b328829e-17ae-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, No, No, Yes, Yes, "",
27581545, 1, 2, 21, 108, Android, -63, 363132, -71, 337368, b328829e-17ae-11eb-adcl-0242ac120002, ele99492-17ae-11eb-adcl-0242ac120002, Yes, Yes, Yes, Yes, "",
68526728, 1, 1, 35, 108, Android, 28, 0262385, -118, 632557, e7bc5fb2-1231-11eb-adcl-0242ac120002, fcbab6aa-1231-11eb-adcl-0242ac120002, Yes, Yes, Yes, Yes, "",

Step3: Created a python file to flatten and clean up the data.(spark_local_flatten.py)

vi spark_local_flatten.py

Explanation of code for spark_local_flatten.py

- Necessary imports

```
import os
import sys
```

- Environment setup

```
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")
```

- Selected imports

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

- Creating a spark session for reading operation

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

- Reading data from HDFS into a data frame for actions to be performed

```
df=spark.read.json("/user/root/clickstream_dump_op/part-00000-10aa7bd4-2598-4e95-a
35d-607379ab389f-c000.json")
```

- Selecting the columns from the clickstream data set from json format to data frame.

```
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")
)
```

- Writing the dataset back to hdfs in csv format for further use case.

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

- Run Spark Submit command

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5
spark_local_flatten.py
```

Step4: Writing script to ingest the relevant bookings data from AWS RDS to Hadoop

- Need to install MySQL connector on EMR

```
wget https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz
tar -xvf mysql-connector-java-8.0.25.tar.gz
```

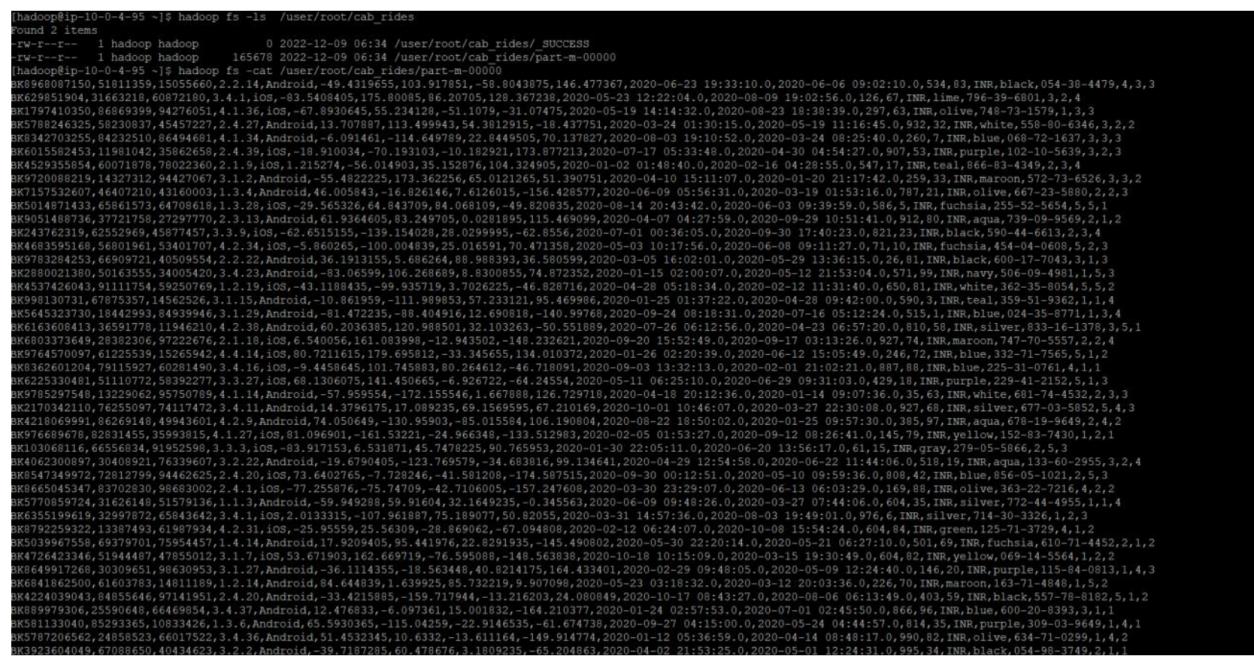
- Running the Sqoop import command to import data from AWS RDS to Hadoop

```
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
```

- Command to view the imported data

```
hadoop fs -cat /user/root/cab_rides/part-m-00000
```

Screenshot showing file contains



```
[hadoop@ip-10-0-4-95 ~]$ hadoop fs -ls /user/root/cab_rides
Found 2 items
-rw-r--r-- 1 hadoop hadoop 0 2022-12-09 06:34 /user/root/cab_rides/_SUCCESS
-rw-r--r-- 1 hadoop hadoop 165678 2022-12-09 06:34 /user/root/cab_rides/part-m-00000
[hadoop@ip-10-0-4-95 ~]$ hadoop fs -cat /user/root/cab_rides/part-m-00000
BK196087150,51811359,15055660,2.2.14,Android,-49.4319655,103.17851,146.477367,2020-06-23 19:33:10.0,2020-06-06 09:02:10.0,534,83,INR,black,054-38-4479,4,3,3
BK229851904,316632180,60872109,3.4,1,108,-83.5408405,155.0005,86.20705,128.367238,2020-05-23 12:22:04.0,2020-08-09 19:02:56.0,126,67,INR,lime,796-39-6801,3,2,4
BK1797410350,86869399,94276051,4,1,36,108,-67.8930645,23.24128,-51.1079,-31.07478,748-73-1579,1,3,3
BK5788246325,58230837,45457227,2.4,2.47,Android,13,707887,13,1499493,54,3812915,-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
BK342702525,84209645,8421490,4,1,34,Android,-0.91461,14.649789,22.8489505,70.137827,2020-08-03 19:10:52.0,2020-03-24 08:25:40.0,260,7,INR,blue,668-72-1637,3,3,3
BK0015582435,11981042,35862658,-4,3,9,108,-10.193103,-10.192921,173.877213,2020-07-17 05:33:48.0,2020-04-30 04:54:27.0,907,53,INR,purple,102-10-5639,3,2,3
BK54959355854,84230934,84231478,-2,1,36,108,-21.3869,-91.89351,38.52878,73.1269,2020-02-01 01:48:00,0,260,6,0,2815,6,1,2,259,3,3,2,3
BK175323607,446072101,41600031,3,4,1,108,-21.3156,-21.28226,-65.0121659,-51.06751,2020-01-21 10:15:20.0,2020-02-02 01:45:20.0,260,6,0,2815,6,1,2,259,3,3,2,3
BK5014871433,65861573,65861573,2,3.1,108,-18.43709,84,068109,-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
BK05014871319,37721750,27297770,2.3,1,13,Android,61.934605,81,249705,0.021891,115,468094,2020-04-07 04:27:59.0,2020-09-29 10:51:41.0,912,80,INR,aqua,739-09-95,2,1,2
BK243762219,62559629,45877676,-3.3,9,108,-62.8655,2020-07-01 00:36:05.0,2020-09-30 17:40:23.0,821,23,INR,black,590-44-4613,2,3,4
BK4603895168,56801961,53401707,4,2,34,108,-5,-860265,-100,004839,25,016591,47,41358,2020-05-03 10:17:56.0,2020-06-08 09:11:27.0,71,10,INR,fuchsia,454-04-0608,5,2,3
BK783284253,66907221,40509554,2,2,22,Android,36,1913155,5,686264,88,988393,36,580598,2020-03-05 16:02:01.0,2020-05-19 20:43:42.0,2020-06-03 09:39:59,0,586,5,INR,fuchsia,255-52-5654,5,5,1
BK2800021388,50163555,34005420,3,4,3,108,-83.06599,106,268681,8,8300855,74,872352,2020-01-15 02:00:07.0,2020-05-12 21:53:04.0,571,99,INR,navy,506-09-4981,1,5,3
BK45174645734,721644570,721644570,2,3.1,108,-43.1188435,-2,1,19,108,-43.1188435,-9.91715,1.21548,-81.93409,141.91351,2020-01-01 01:48:00,0,260,6,0,2815,6,1,2,259,3,2,3
BK991130731,67875357,14562526,3,1,15,Android,-10.181959,-11,9898353,57,233121,95,469984,2020-01-25 01:37:22.0,2020-04-28 09:42:00.0,590,3,INR,teal,359-51-9362,1,1,4
BK5645232730,84230934,84231478,3,1,1,108,-18.43709,84,068109,-49.820835,2020-09-24 08:18:31.0,2020-07-16 05:12:24.0,515,1,1,INR,blue,624-35-8771,1,3,4
BK1613608413,36591778,11946210,4,2,38,Android,60,2036385,120,985051,32,102363,50,551889,2020-07-26 06:12:56.0,2020-04-23 06:57:20.0,810,58,INR,silver,833-16-1378,3,5,1
BK603373649,28382306,97222676,2,1,18,108,-6.540056,161,018398,-12,943502,-14.232621,2020-09-20 15:52:49.0,2020-09-17 03:13:26.0,161,74,INR,maroon,747-70-5557,2,2,4
BK3764570097,61225539,15265942,4,2,14,108,-83.7211615,179,659812,-33,354655,143,010372,2020-01-26 02:20:39.0,2020-06-12 15:05:49.0,246,72,INR,blue,332-71-7565,5,1,2
BK362601204,60281490,20281490,3,4,16,108,-3,-445645,101,745883,80,264612,-46,718091,2020-09-03 13:32:13.0,2020-02-01 21:20:39.0,2020-05-01 06:25:10.0,2020-06-29 09:31:03.0,0,429,18,INR,blue,225-31-0761,4,1,1
BK225330481,5111072,5339227,3,3,27,108,68,1306075,141,450465,-92,926722,-64,245544,2020-05-11 06:25:10.0,2020-06-29 09:31:03.0,0,429,18,INR,purple,229-41-2152,5,1,3
BK785297548,13229062,95290569,1,4,14,Android,-17,9250769,1.6778988,126,729718,2020-04-10 20:13:36.0,2020-01-14 09:07:36.0,35,63,INR,white,681-74-4532,2,2,3
BK2170324110,76255097,74117472,3,4,11,Android,14,3796175,17,089233,69,1569895,67,210169,2020-10-01 10:46:07.0,2020-03-27 22:30:00.0,927,68,INR,silver,677-03-5852,5,4,3
BK4218069911,86269148,49943601,4,2,9,Android,74,050649,-16,095903,-13.501584,106,198080,2020-08-02 09:00:00.0,2020-01-25 09:57:30.0,385,97,INR,aqua,678-19-9649,2,4,2
BK76669478,82831455,35993815,4,1,27,108,-18.096901,-161,5129321,2020-09-12 08:26:41.0,145,79,INR,yellow,152-83-7430,1,2,1
BK103098116,66556384,9195298,3,3,3,108,-83.91753,5,63,6598171,45,7478229,2020-01-30 22:05:11.0,2020-06-20 13:56:17.0,61,0,INR,gray,279-05-5866,2,5,3
BK462300897,30408921,76339607,3,2,22,Android,-19,6790405,12,76,769579,-34,683916,99,134461,2020-04-29 12:54:58.0,2020-06-22 11:44:06,0,519,19,INR,aqua,133-60-2955,3,2,4
BK54743493,7202799,94462629,-4,20,108,-73,6402765,-17,728246,-41,581208,-47,710605,157,24,160833,2020-03-30 23:49:00.0,2020-06-17 06:03:29.0,169,88,INR,olive,363-31-2164,4,2,2
BK375722682,31629148,51571369,4,2,27,108,-83.91753,5,63,6598171,45,7478229,2020-03-30 17:37:36.0,2020-06-16 04:21:40.0,25,9455,1,1,4
BK5385199619,55984253,55984253,3,4,11,108,21,213315,-107,916177,75,1389077,50,82055,2020-03-28 13:45:34.0,2020-09-03 19:42:01.0,576,8,INR,red,714-30-3226,5,1,2
BK792259322,13297893,6199734,1.6899734,2,3,1,108,-25,85559,25,536039,-28,869062,-17,094806,2020-02-12 06:24:07.0,2020-10-08 15:54:24.0,604,84,INR,green,125-71-3729,4,1,2
BK50340627555,69379701,97595457,1,4,14,Android,17,9209405,95,499179,-125,22,8291935,-145,490802,2020-05-30 22:20:14,0,2020-05-21 06:27:10.0,501,69,INR,fuchsia,610-71-4452,2,1,2
BK4726423346,51944479,47855012,3,1,17,108,53,671903,162,669719,-76,595838,2020-10-18 10:15:09,0,2020-03-19:30:49.0,604,82,INR,yellow,069-14-5564,1,2,2
BK36049017268,30309651,85293365,1,3,6,Android,-18,563444,40,8214175,163,433401,2020-02-29 09:48:00.0,2020-05-30 12:24:40.0,146,20,INR,purple,115-84-0813,1,4,3
BK56841862500,61603783,1481111891,2,1,Android,84,644839,1,639925,85,732219,97,907998,2020-05-23 03:18:32.0,2020-03-12 20:03:36.0,226,70,INR,maroon,163-71-4848,1,5,2
BK224201076,84855646,97141951,2,4,20,Android,-33,4215889,-17,717944,-13,216203,24,080849,2020-10-17 08:43:27.0,2020-08-06 06:13:49.0,403,59,INR,black,557-78-8182,5,1,2
BK58997306,25590648,66469854,3,4,37,Android,12,476833,-6,097361,155,001832,-164,210377,2020-01-24 02:57:53,0,2020-07-01 02:45:50,0,866,96,INR,blue,220-20-8393,3,1,1
BK58113340,84230934,84231478,2,3,1,108,-18,653444,40,8214175,163,433401,2020-02-29 09:48:00.0,2020-05-30 12:24:40.0,146,20,INR,purple,115-84-0813,1,4,2
BK57872026562,2485823,66017522,3,4,3,6,Android,51,5132435,10,6332,-16,111614,-149,914774,2020-01-12 05:36:59,0,2020-04-14 08:48:17.0,992,82,INR,olive,634-71-0299,1,4,2
BK592360409,67088650,40434623,3,2,2,Android,-39,7187285,6,478676,3,1809235,-65,204863,2020-04-02 21:53:25,0,995,34,INR,black,054-98-3749,2,1,1
```

Step5: Creating aggregates for finding date-wise total bookings using the Spark script

- Created a python file which will contain the code to aggregate the booking data for finding date wise total bookings.

```
vi datewise_bookings_aggregates_spark.py
```

Explanation of code for datewise_bookings_aggregates_spark.py

- Necessary imports

```
import os
```

```
import sys
```

- Environment setup

```
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")
```

- Selective imports

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

- Creating spark session for reading operation of data into data frame

```
spark=SparkSession.builder.appName("datewise_bookings_aggregates_spark").master(
"local").getOrCreate()
```

- Reading data from HDFS to data frame for actions to be performed

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

- Getting The count of the dataset

```
df.count()
```

- Renaming some columns for better readability and understanding

```
new_col =
["booking_id","customer_id","driver_id","customer_app_version","customer_phone_os_v
ersion","pickup_lat","pickup_lon","drop_lat",
"drop_lon","pickup_timestamp","drop_timestamp","trip_fare","tip_amount","currency_cod
e","cab_color","cab_registration_no","customer_rating_by_driver",
"rating_by_customer","passenger_count"]
```

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

- 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")
```

- Aggregating data on pickup_date

```
agg_df=new_df.groupBy("pickup_date").count().orderBy("pickup_date")
```

- The count of Bookings aggregates_table

```
agg_df.count()
```

- Writing aggregated data to HDFS into CSV file format

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

Screenshot showing files created in HDFS

```
[hadoop@ip-10-0-4-95 ~]$ hadoop fs -ls /user/root/datewise_bookings_agg
Found 2 items
-rw-r--r-- 1 hadoop hadoop 0 2022-12-09 12:45 /user/root/datewise_bookings_agg/_SUCCESS
-rw-r--r-- 1 hadoop hadoop 3776 2022-12-09 12:45 /user/root/datewise_bookings_agg/part-00000-58cdf4a6-2249-42d5-bc65-b312c5d39846-c000.csv
[hadoop@ip-10-0-4-95 ~]$ hadoop fs -cat /user/root/datewise_bookings_agg/part-00000-58cdf4a6-2249-42d5-bc65-b312c5d39846-c000.csv
pickup_date,count
2020-01-01,1
2020-01-02,3
2020-01-03,2
2020-01-04,2
2020-01-05,2
2020-01-06,3
2020-01-07,2
2020-01-08,4
2020-01-09,2
2020-01-10,2
2020-01-11,3
2020-01-12,3
2020-01-13,2
2020-01-14,2
2020-01-15,5
2020-01-16,3
2020-01-17,4
2020-01-18,4
2020-01-20,4
2020-01-21,1
2020-01-23,4
2020-01-24,8
2020-01-25,4
2020-01-26,4
2020-01-27,3
2020-01-28,9
2020-01-29,2
2020-01-30,4
2020-01-31,5
2020-02-01,7
2020-02-02,6
2020-02-03,3
2020-02-04,4
2020-02-05,3
2020-02-06,3
2020-02-07,3
2020-02-08,6
2020-02-09,6
2020-02-10,3
2020-02-11,1
2020-02-12,3
2020-02-13,2
2020-02-15,3
2020-02-16,3
2020-02-17,10
```

Step6: Creating database and tables in hive and performing the tasks.

- **Creating Hive Managed Tables to perform use case queries.**

First create a database

```
create database if not exists cab_booking_data ;
use cab_booking_data;
```

Create **Clickstream** 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;
```

Screenshot showing clickstream table created

```
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: 0.547 seconds
```

Create Booking table

```

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;
  
```

Screenshot showing booking table created

```

Time taken: 0.347 seconds
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: 0.105 seconds
  
```

Create Aggregated date wise table

```

CREATE TABLE IF NOT EXISTS aggregate_datewise
(
    pickup_date      DATE,
    booking_id_count INT
  
```

```
) row format delimited fields TERMINATED BY ',' stored AS textfile;
```

Screenshot showing aggregated date wise table created

- Commands to load the data into Hive tables

1. Loading data into clickstream table

```
load data local inpath '/home/hadoop/clickstream/clickstream_flattened.csv' into table
clickstream;
```

2. Loading data into booking table

```
load data local inpath '/home/hadoop/cab_rides/part-m-00000' into table booking;
```

3. Loading data into aggregate_datewise table

```
load data local inpath '/home/hadoop/datewise_bookings_agg/datewise_agg' into table
aggregate_datewise;
```

Screenshot showing loading of data into different tables

```
Time taken: 1.857 seconds, Fetched: 1000 row(s)
hive> load data local inpath '/home/hadoop/clickstream/clickstream_flattened.csv' into table clickstream;
Loading data to table capstone.clickstream
OK
Time taken: 0.442 seconds
```

```
hive> load data local inpath '/home/hadoop/cab_rides/part-m-00000' into table booking;
Loading data to table capstone.booking
OK
Time taken: 1.566 seconds
hive> select * from capstone.booking;
OK
```

```
FAILED: SemanticException Line 1:3 invalid path '/home/hadoop/datewise_bookings_agg/datewise_agg'. No files matched
hive> load data local inpath '/home/hadoop/datewise_bookings_agg/datewise_agg.csv' into table aggregate_datewise;
Loading data to table capstone.aggregate_datewise
OK
Time taken: 0.426 seconds
```