

## Scripts Execution

### Screenshots of the execution of the scripts written

- Loading member score data stored in central AWS RDS:

```
memberschema = StructType([StructField('member_id', StringType(),False),
                                StructField('score', IntegerType(),False),
                                ])

memf = spark.read.csv("hdfs://user/root/cap_project/member_score", header = False, schema = memberschema)

memf.show()
```

member_id	score
000037495066290	339
000117826301530	289
001147922084344	393
001314074991813	225
001739553947511	642
003761426295463	413
004494068832701	217
006836124210484	504
006991872634058	697
007955566230397	372
008732267588672	213

- Loading the card holder's data stored in central AWS RDS:

```
cardschema = StructType([StructField('card_id', StringType(),False),
                                StructField('member_id', StringType(),False),
                                StructField('member_joining_dt', StringType(),False),
                                StructField('card_purchase_dt', StringType(),False),
                                StructField('country', StringType(),False),
                                StructField('city', StringType(),False),
                                ])

cardf = spark.read.csv("hdfs://user/root/cap_project/card_member", header = False, schema = cardschema)

cardf.show()
```

card_id	member_id	member_joining_dt	card_purchase_dt	country	city
340028465709212	009250698176266	2012-02-08 06:04:...	05/13	United States	Barberton
340054675199675	835873341185231	2017-03-10 09:24:...	03/17	United States	Fort Dodge
340082915339645	512969555857346	2014-02-15 06:30:...	07/14	United States	Graham
340134186926007	887711945571282	2012-02-05 01:21:...	02/13	United States	Dix Hills
340265728490548	680324265406190	2014-03-29 07:49:...	11/14	United States	Rancho Cucamonga
340268219434811	929799084911715	2012-07-08 02:46:...	08/12	United States	San Francisco
340379737226464	089615510858348	2010-03-10 00:06:...	09/10	United States	Clinton
340383645652108	181180599313885	2012-02-24 05:32:...	10/16	United States	West New York
340803866934451	417664728506297	2015-05-21 04:30:...	08/17	United States	Beaverton

- Loading card\_transaction data from csv:

```
transacstion = StructType([StructField('card_id', StringType(),False),
                             StructField('member_id', StringType(),False),
                             StructField('amount', IntegerType(),False),
                             StructField('postcode', StringType(),False),
                             StructField('pos_id', StringType(),False),
                             StructField('transaction_dt', StringType(),False),
                             StructField('status', StringType(),False),
                             ])
```

```
tranf = spark.read.csv("hdfs://user/root/cap_project/card_transactions.csv", header = True, schema = transacstion)
```

```
tranf= tranf.filter(tranf.status!='FRAUD')
```

```
tranf.show()
```

card_id	member_id	amount	postcode	pos_id	transaction_dt	status
348702330256514	000037495066290	9084849	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	330148	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	136052	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	4310362	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	9097094	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	2291118	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	4900011	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	633447	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	6259303	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	369067	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	1193207	33946	614677375609919	11-02-2018 00:00:00	GENUINE

- Joining the member\_score and card\_member on member\_id to extract credit score of each member and selecting the required fields:

```
score = memf.join(cardf, memf.mem_id == cardf.member_id,how='LEFT')
```

```
score.printSchema()
```

```
root
|-- mem_id: string (nullable = true)
|-- score: integer (nullable = true)
|-- card_id: string (nullable = true)
|-- member_id: string (nullable = true)
|-- member_joining_dt: string (nullable = true)
|-- card_purchase_dt: string (nullable = true)
|-- country: string (nullable = true)
|-- city: string (nullable = true)
```

```
score = score.select('mem_id', 'score', 'card_id')
```

```
score.show()
```

mem_id	score	card_id
000037495066290	339	348702330256514
000117826301530	289	5189563368503974
001147922084344	393	5407073344486464
001314074991813	225	378303738095292
001739553947511	642	348413196172048
003761426295463	413	348536585266345
004494068832701	217	5515987071565183

- Joining both the history transactions CSV and score DF:

```
hist = tranf.join(score, tranf.member_id == score.mem_id,how='outer')
```

```
hist.printSchema()
```

```
root
|-- card_id: string (nullable = true)
|-- member_id: string (nullable = true)
|-- amount: integer (nullable = true)
|-- postcode: string (nullable = true)
|-- pos_id: string (nullable = true)
|-- transaction_dt: string (nullable = true)
|-- status: string (nullable = true)
|-- mem_id: string (nullable = true)
|-- score: integer (nullable = true)
|-- cardid: string (nullable = true)
```

```
hist = hist.select('card_id', 'amount', 'postcode', 'pos_id','transaction_dt','status','score')
```

```
hist.show()
```

card_id	amount	postcode	pos_id	transaction_dt	status	score
340379737226464	6126197	46933	167473544283898	01-05-2016 08:10:50	GENUINE	229
340379737226464	7949232	61840	664980919335952	01-10-2016 10:38:52	GENUINE	229
340379737226464	943839	91743	633038040069180	02-08-2016 00:31:25	GENUINE	229
340379737226464	3764114	91743	633038040069180	02-08-2016 21:35:27	GENUINE	229
340379737226464	6221251	98384	064948657945290	02-10-2016 14:44:14	GENUINE	229
340379737226464	2868312	26032	856772774421259	02-12-2016 21:55:43	GENUINE	229
340379737226464	4418586	20129	390339673634463	02-12-2017 17:05:51	GENUINE	229

Compute the max transaction date:

```
look_up_table = history.groupBy('card_id').agg(f.max("transaction_date").alias('transaction_date'))
```

```
look_up_table.show()
```

card_id	transaction_date
340379737226464	2018-01-27 00:19:47
377201318164757	2017-11-28 16:32:22
348962542187595	2018-01-29 17:17:14
4389973676463558	2018-01-26 13:47:46
5403923427969691	2018-01-22 23:46:19
345406224887566	2017-12-25 04:03:58
6562510549485881	2018-01-17 08:35:27
5508842242491554	2018-01-31 14:55:58
4407230633003235	2018-01-27 07:21:08
379321864695232	2018-01-03 00:29:37

Inner join on look up table dataset on card\_id:

```
look_up_table = look_up_table.join(score, look_up_table.card_id == score.cardid,how='INNER')
```

```
look_up_table.show()
```

card_id	transaction_date	mem_id	score	cardid
340379737226464	2018-01-27 00:19:47	089615510858348	229	340379737226464
345406224887566	2017-12-25 04:03:58	296206661780881	349	345406224887566
348962542187595	2018-01-29 17:17:14	366246487993992	522	348962542187595
377201318164757	2017-11-28 16:32:22	924475891017022	432	377201318164757



- Calculate the Upper Control Limit (UCL),  $UCL = \text{Moving Average} + 3 * (\text{Standard Deviation})$ , We shall first calculate the moving average of card amount for last 10 transactions.
- Create a window over existing dataframe and aggregate the same card\_id, the dataframe is grouped by card\_id and then order by transaction\_date.

```

window = Window.partitionBy(history['card_id']).orderBy(history['transaction_date'].desc())
history_df = history.select('*', f.rank().over(window).alias('rank')).filter(f.col('rank') <= 10)

history_df.show()

```

card_id	amount	postcode	pos_id	status	score	transaction_date	rank
340379737226464	1784098	26656	000383013889790	GENUINE	229	2018-01-27 00:19:47	1
340379737226464	3759577	61334	016312401940277	GENUINE	229	2018-01-18 14:26:09	2
340379737226464	4080612	51338	562082278231631	GENUINE	229	2018-01-14 20:54:02	3
340379737226464	4242710	96105	285501971776349	GENUINE	229	2018-01-11 19:09:55	4
340379737226464	9061517	40932	232455833079472	GENUINE	229	2018-01-10 20:20:33	5
340379737226464	102248	40932	232455833079472	GENUINE	229	2018-01-10 15:04:33	6
340379737226464	7445128	50455	915439934619047	GENUINE	229	2018-01-07 23:52:27	7
340379737226464	5706163	50455	915439934619047	GENUINE	229	2018-01-07 22:07:07	8
340379737226464	8090127	18626	359283931604637	GENUINE	229	2017-12-29 13:24:07	9
340379737226464	9282351	41859	808326141065551	GENUINE	229	2017-12-28 19:50:46	10

- To import all SQL functions to pyspark, we need to import the necessary functions (import pyspark.sql.functions as f)

```

history_df = history_df.groupBy("card_id").agg(f.round(f.avg('amount'),2).alias('moving_avg'), \
                                              f.round(f.stddev('amount'),2).alias('Std_Dev'))

```

card_id	moving_avg	Std_Dev
340379737226464	5355453.1	3107063.55
345406224887566	5488456.5	3252527.52
348962542187595	5735629.0	3089916.54
377201318164757	5742377.7	2768545.84
379321864695232	4713319.1	3203114.94
4389973676463558	4923904.7	2306771.9
4407230633003235	4348891.3	3274883.95

Calculate UCL from the computed standard deviation and moving average:

```

history_df = history_df.withColumn('UCL', history_df.moving_avg + 3 * (history_df.Std_Dev))

```

card_id	moving_avg	Std_Dev	UCL
340379737226464	5355453.1	3107063.55	1.4676643749999998E7
345406224887566	5488456.5	3252527.52	1.524603906E7
348962542187595	5735629.0	3089916.54	1.5005378620000001E7
377201318164757	5742377.7	2768545.84	1.4048015219999999E7
379321864695232	4713319.1	3203114.94	1.432266392E7
4389973676463558	4923904.7	2306771.9	1.1844220399999999E7
4407230633003235	4348891.3	3274883.95	1.4173543150000002E7

Joining the dataframe with previous dataframe on card\_id:

```
history_df = history_df.select('card_id', 'UCL')
```

```
look_up_table = look_up_table.join(history_df, on=['card_id'])
```

```
look_up_table.show()
```

card_id	transaction_date	score	postcode	UCL
340379737226464	2018-01-27 00:19:47	229	26656	1.4676643749999998E7
345406224887566	2017-12-25 04:03:58	349	53034	1.524603906E7
348962542187595	2018-01-29 17:17:14	522	27830	1.5005378620000001E7
377201318164757	2017-11-28 16:32:22	432	84302	1.4048015219999999E7
379321864695232	2018-01-03 00:29:37	297	98837	1.432266392E7
4389973676463558	2018-01-26 13:47:46	400	10985	1.1844220399999999E7
4407230633003235	2018-01-27 07:21:08	567	50167	1.4173543150000002E7
5403923427969691	2018-01-22 23:46:19	324	17350	1.411602776E7

Remove duplicate on redundant transactions done on card\_id, transaction\_date and postcode:

```
look_up_table = look_up_table.dropDuplicates(['card_id', 'transaction_date', 'postcode'])
```

```
look_up_table.count()
```

```
1000
```

1. Load the dataframe into the look up table, happybase API shall be used. 1<sup>st</sup> step is to connect with hbase:

```
import happybase
#To create connection
connection = happybase.Connection('localhost', port=9090, autoconnect=False)
```

```
def open_connection():
    connection.open()
#To close the opened connection
def close_connection():
    connection.close()
#To list all tables in Hbase
def list_tables():
    print "fetching all table"
    open_connection()
    tables = connection.tables()
    close_connection()
    print "all tables fetched"
    return tables
```

```
#To create the required table
def create_table(name, cf):
    print "creating table " + name
    tables = list_tables()
    if name not in tables:
        open_connection()
        connection.create_table(name, cf)
        close_connection()
        print "table created"
    else:
        print "table already present"
#To get the table created
def get_table(name):
    open_connection()
    table = connection.table(name)
    close_connection()
    return table
```

## 2. If table does not exist, create the table :

```
create_table('look_up_table', {'info' : dict(max_versions=5) })
```

```
creating table look_up_table
fetching all table
all tables fetched
table created
```

## 3. Batch insert data into the table:

```
#To batch insert data in lookup table
def batch_insert_data(df,tableName):
    print "starting batch insert of events"
    table = get_table(tableName)
    open_connection()
    rows_count=0

#To Create a rowkey for better data query
    rowKey_dict={}
    with table.batch(batch_size=4) as b:
        for row in df.rdd.collect():
            b.put(bytes(row.card_id), { 'info:card_id':bytes(row.card_id),
                                         'info:transaction_date':bytes(row.transaction_date),
                                         'info:score':bytes(row.score),
                                         'info:postcode':bytes(row.postcode),
                                         'info:UCL':bytes(row.UCL)})

    print "batch insert done"
    close_connection()
```

```
batch_insert_data(look_up_table,'look_up_table')
```

```
starting batch insert of events
batch insert done
```

## 4. Once the batch insertion is complete, login to putty as root user and enter Hbase shell.

```
5232083808576685 column=info:card_id, timestamp=1607880086427, value=5232083808576685
5232083808576685 column=info:postcode, timestamp=1607880086427, value=17965
5232083808576685 column=info:score, timestamp=1607880086427, value=566
5232083808576685 column=info:transaction_date, timestamp=1607880086427, value=2018-01-09 12:44:31
5232271306465150 column=info:UCL, timestamp=1607880087122, value=10951781.35
5232271306465150 column=info:card_id, timestamp=1607880087122, value=5232271306465150
5232271306465150 column=info:postcode, timestamp=1607880087122, value=12920
5232271306465150 column=info:score, timestamp=1607880087122, value=638
5232271306465150 column=info:transaction_date, timestamp=1607880087122, value=2018-01-22 16:44:59
5232695950818720 column=info:UCL, timestamp=1607880087849, value=15220850.52
5232695950818720 column=info:card_id, timestamp=1607880087849, value=5232695950818720
5232695950818720 column=info:postcode, timestamp=1607880087849, value=79080
5232695950818720 column=info:score, timestamp=1607880087849, value=207
5232695950818720 column=info:transaction_date, timestamp=1607880087849, value=2018-01-29 08:30:32
5239380866598772 column=info:UCL, timestamp=1607880086358, value=12835247.22
5239380866598772 column=info:card_id, timestamp=1607880086358, value=5239380866598772
5239380866598772 column=info:postcode, timestamp=1607880086358, value=72471
5239380866598772 column=info:score, timestamp=1607880086358, value=440
5239380866598772 column=info:transaction_date, timestamp=1607880086358, value=2017-12-07 21:44:43
5242841712000086 column=info:UCL, timestamp=1607880088013, value=15646358.41
5242841712000086 column=info:card_id, timestamp=1607880088013, value=5242841712000086
5242841712000086 column=info:postCode, timestamp=1607880088013, value=48821
5242841712000086 column=info:score, timestamp=1607880088013, value=236
5242841712000086 column=info:transaction_date, timestamp=1607880088013, value=2018-01-27 10:51:48
5249623960609831 column=info:UCL, timestamp=1607880087191, value=12497504.76
5249623960609831 column=info:card_id, timestamp=1607880087191, value=5249623960609831
5249623960609831 column=info:postCode, timestamp=1607880087191, value=16858
5249623960609831 column=info:score, timestamp=1607880087191, value=265
5249623960609831 column=info:transaction_date, timestamp=1607880087191, value=2018-01-28 00:54:29
5252551880815473 column=info:UCL, timestamp=1607880086480, value=11540779.75
5252551880815473 column=info:card_id, timestamp=1607880086480, value=5252551880815473
5252551880815473 column=info:postCode, timestamp=1607880086480, value=39352
5252551880815473 column=info:score, timestamp=1607880086480, value=449
5252551880815473 column=info:transaction_date, timestamp=1607880086480, value=2018-02-01 10:14:39
5253084214148600 column=info:UCL, timestamp=1607880087349, value=13198338.6
5253084214148600 column=info:card_id, timestamp=1607880087349, value=5253084214148600
5253084214148600 column=info:postCode, timestamp=1607880087349, value=78054
5253084214148600 column=info:score, timestamp=1607880087349, value=512
5253084214148600 column=info:transaction_date, timestamp=1607880087349, value=2018-01-27 10:51:49
5254025009868430 column=info:UCL, timestamp=1607880087698, value=14556419.87
```