

# Scripts Execution

## Explanation of the solution to the batch layer problem

- 1) Solution to the current problem statement is developed on Pyspark.
- 2) As 1<sup>st</sup> step, we need to load data which is present in RDS to HDFS using Sqoop import commands.

Table 1 (member\_score)

```
sqoop import \  
--connect jdbc:mysql://upgradawsrds1.cyaielc9bmnf.us-east-1.rds.amazonaws.com/cred_financials_data \  
--table member_score \  
--username upgraduser --password upgraduser \  
--target-dir /user/root/cap_project/member_score \  
-m 1
```

Table 2 (card\_member)

```
sqoop import \  
--connect jdbc:mysql://upgradawsrds1.cyaielc9bmnf.us-east-1.rds.amazonaws.com/cred_financials_data \  
--table card_member \  
--username upgraduser --password upgraduser \  
--target-dir /user/root/cap_project/card_member \  
-m 1
```

- 3) Load card\_transactions.csv to HDFS after moving it to EC2-USER by using command  
`hadoop fs -copyFromLocal /home/ec2-user/card_transaction.csv cap_project/card_transaction.csv`
- 4) Now, connect to putty instance and load jupyter notebook from root user, by using command  
`jupyter notebook --port 7861 --allow-root`
- 5) Open a new notebook and load a spark context.
- 6) Start reading all 3 files namely CARD\_MEMBER, MEMBER\_SCORE & CARD\_TRANSACTIONS in Pyspark notebook into predefined file schemas.
- 7) Once read is successful, put a command `df.show()` to see data is read successfully.

```
In [9]: 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),
                                   ])
```

```
In [10]: 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
340889618969736	459292914761635	2013-04-23 08:40:...	11/15	United States	West Palm Beach
340924125838453	188119365574843	2011-04-12 04:28:...	12/13	United States	Scottsbluff
341005627432127	872138964937565	2013-09-08 03:16:...	02/17	United States	Chillum
341029651579925	974087224071871	2011-01-14 00:20:...	08/12	United States	Valley Station
341311317050937	561687420200207	2014-03-18 06:23:...	02/15	United States	Vincennes
341344252914274	695906467918552	2012-03-02 03:21:...	03/13	United States	Columbine
341363858179050	009190444424572	2012-02-19 05:16:...	04/14	United States	Cheektowaga
341519629171378	533670008048847	2013-05-13 07:59:...	01/15	United States	Centennial
341641153427489	230523184584316	2013-03-25 08:51:...	11/15	United States	Colchester
341719092861087	304847505155781	2015-12-06 08:06:...	11/17	United States	Vernon Hills
341722035429601	979218131207765	2015-12-22 10:46:...	01/17	United States	Elk Grove Village

only showing top 20 rows

Activate Windows  
Go to Settings to activate

```
In [17]: memberschema = StructType([StructField('member_id', StringType(),False),
                                       StructField('score', IntegerType(),False),
                                       ])
```

```
In [18]: memf = spark.read.csv("hdfs://user/root/cap_project/member_score", header = False, schema = memberschema)
```

```
In [19]: memf.count()
```

```
Out[19]: 999
```

```
In [20]: 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
008765307152821	399

Activate Windows  
Go to Settings to activate

```
In [24]: transaction = 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),
    ])
```

```
In [25]: tranf = spark.read.csv("hdfs://user/root/cap_project/card_transactions.csv", header = True, schema = transaction)
```

```
In [26]: tranf.count()
```

```
Out[26]: 53292
```

```
In [29]: 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
348702330256514	000037495066290	9335696	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	2241736	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	457701	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	7176668	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	5585098	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	7918756	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	1611089	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	217221	33946	614677375609919	11-02-2018 00:00:00	GENUINE
348702330256514	000037495066290	2617991	33946	614677375609919	11-02-2018 00:00:00	GENUINE

only showing top 20 rows

- Once we load all input data, next we need to join all these files to form an amalgam and extract only relevant fields out of them that are need for our analysis.
- First join card\_member & member\_score dataframes and slide credit score into card\_member by using member\_id field as join key

```
In [30]: score = memf.join(cardf, memf.mem_id == cardf.member_id,how='LEFT')
```

```
In [31]: score.count()
```

```
Out[31]: 999
```

```
In [32]: 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)
```

- 10) With the fresh dataframe, use member ID once again as common key and join with card\_transaction.csv to load postcode, pos\_id, status, amount & transaction date fields from history transactions.
- 11) To arrive at derived columns like latest\_transaction date, group the combined dataframe on card\_id such that all transactions on same card id collate and get max(transaction date). Append this column to combined dataframe.

```
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|
| 340028465709212| 2018-01-02 03:25:35|
| 349143706735646| 2018-01-29 22:33:14|
| 4126356979547079| 2018-01-24 16:09:03|
| 5543219113990484| 2018-01-13 18:34:00|
| 5464688416792307| 2018-01-26 19:03:47|
| 6011273561157733| 2018-02-01 01:27:58|
| 4484950467600170| 2018-01-10 08:03:13|
| 4818950814628962| 2018-01-31 00:53:15|
| 5573293264792992| 2018-01-31 14:55:57|
| 6011985140563103| 2018-01-30 02:03:54|
+-----+
```

Activate Window:  
Go to Settings to activate

```
In [63]: look_up_table = look_up_table.join(history,on=['card_id','transaction_date'],how='left')
look_up_table.count()
```

```
Out[63]: 1476
```

```
In [64]: look_up_table.show()
```

```
+-----+-----+-----+-----+-----+-----+
|      card_id| transaction_date| amount|postcode|      pos_id| status|score|
+-----+-----+-----+-----+-----+-----+
| 378586484293754| 2017-12-24 05:14:37| 3859271| 24363| 753115024049849| GENUINE| 337|
| 4356201405998945| 2018-01-24 14:23:42| 4553231| 43791| 339439168301190| GENUINE| 600|
| 4418227862530505| 2018-01-25 16:43:45| 4085014| 14544| 028630406062180| GENUINE| 318|
| 5400249950855567| 2018-01-28 06:10:31| 1062269| 24966| 757227694469394| GENUINE| 523|
| 373748808330229| 2018-01-29 13:46:32| 2446006| 25260| 459926365561014| GENUINE| 685|
| 4353614029446427| 2018-01-10 23:51:13| 2713094| 15311| 791335648163958| GENUINE| 219|
| 4598225659063187| 2018-01-25 21:59:45| 421272| 50531| 657401894365206| GENUINE| 355|
| 4689314809377828| 2018-01-25 21:59:45| 1151530| 29550| 365821079545471| GENUINE| 632|
| 5447036761675606| 2017-11-16 23:38:38| 566003| 32970| 900066068310939| GENUINE| 690|
| 5508842242491554| 2018-01-31 14:55:58| 2710473| 12986| 990193545769550| GENUINE| 585|
| 5572427538311236| 2018-01-31 20:11:58| 2479113| 91040| 341079781603709| GENUINE| 303|
| 6011654527329500| 2018-01-31 00:53:16| 9773537| 58634| 018255965744212| GENUINE| 683|
| 347893423075811| 2018-01-24 02:06:21| 2927191| 15532| 320818315059172| GENUINE| 429|
| 371085417506954| 2018-01-28 14:57:11| 741464| 19468| 551815269280261| GENUINE| 599|
| 5316831626197194| 2018-01-29 13:46:32| 6716709| 40488| 704363694703346| GENUINE| 227|
```

Activate Windows  
Go to Settings to activate

- 12) Now, calculate the UCL value that mainly revolves around "amount" field. We all know UCL can be calculated as moving average + 3 \*(standard deviation). Hence we open a window frame where we group input dataframe rows on card\_id and order by transaction date to get all transactions on card in chronological order.
- 13) Now, once you group & order by transactions rank these chronological transactions starting from 1 till go on.
- 14) Pick rows only whose rank is less than 10, by which we select moving average of top 10 latest transactions done on card\_id.

- 15) Import SQL functions library in pyspark and perform avg() function on top 10 rows of grouped card\_id.
- 16) Similarly, perform stddev() to derive standard deviation on these top 10 rows selected by rank.
- 17) Perform computation as per formula given to deduce UCL value and append this to original dataframe obtained at step 11.

```
In [67]: 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)
```

```
In [68]: 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
345406224887566	1135534	53034	146838238062262	GENUINE	349	2017-12-25 04:03:58	1
345406224887566	5190295	88036	821406924682103	GENUINE	349	2017-12-20 04:41:07	2
345406224887566	5970187	28334	024341862357645	GENUINE	349	2017-11-30 05:24:25	3
345406224887566	3854486	48880	172521878612232	GENUINE	349	2017-09-21 00:01:58	4

Activate Windows  
Go to Settings to activate

```
In [69]: 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'))
history_df.show()
```

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
5403923427969691	5375495.6	2913510.72
5508842242491554	4570725.9	3229905.04
6562510549485881	5551056.9	2501552.48
340028465709212	6863758.9	3326644.65
349143706735646	5453372.9	3424332.26
4126356979547079	4286400.2	2909676.26
4484950467600170	4550480.5	3171538.48
4818950814628962	2210428.9	958307.87
5464688416792307	4985938.2	2379084.95
5543219113990484	4033586.9	2969107.42
5573293264792992	3929994.0	2589503.93
6011273561157733	4634624.8	2801886.17
6011985140563103	5302878.9	3088988.7

Activate Windows  
Go to Settings to activate



```
In [70]: history_df = history_df.withColumn('UCL', history_df.moving_avg + 3 * (history_df.Std_Dev))
history_df.show()
```

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
5403923427969691	5375495.6	2913510.72	1.411602776E7
5508842242491554	4570725.9	3229905.04	1.4260441020000001E7
6562510549485881	5551056.9	2501552.48	1.305571434E7
340028465709212	6863758.9	3326644.65	1.684369285E7
349143706735646	5453372.9	3424332.26	1.572636968E7
4126356979547079	4286400.2	2909676.26	1.301542898E7
4484950467600170	4550480.5	3171538.48	1.406509594E7
4818950814628962	2210428.9	958307.87	5085352.51
5464688416792307	4985938.2	2379084.95	1.212319305E7
5543219113990484	4033586.9	2969107.42	1.294090916E7
5573293264792992	3929994.0	2589503.93	1.1698505790000001E7
6011273561157733	4634624.8	2801886.17	1.3040283309999999E7
6011985140563103	5302878.9	3088988.7	1.4569845000000002E7

Activate Window  
Go to Settings to activate

18) Final dataset looks like –

```
In [72]: look_up_table = look_up_table.join(history_df, on=['card_id'])
```

```
In [73]: 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
5508842242491554	2018-01-31 14:55:58	585	12986	1.4260441020000001E7
6562510549485881	2018-01-17 08:35:27	518	35440	1.305571434E7
340028465709212	2018-01-02 03:25:35	233	24658	1.684369285E7
349143706735646	2018-01-29 22:33:14	298	99101	1.572636968E7
4126356979547079	2018-01-24 16:09:03	345	14475	1.301542898E7
4484950467600170	2018-01-10 08:03:13	462	13324	1.406509594E7
4818950814628962	2018-01-31 00:53:15	660	88081	5085352.51
5464688416792307	2018-01-26 19:03:47	469	71670	1.212319305E7
5543219113990484	2018-01-13 18:34:00	494	62273	1.294090916E7
5573293264792992	2018-01-31 14:55:57	284	27012	1.1698505790000001E7
6011273561157733	2018-02-01 01:27:58	411	45305	1.3040283309999999E7
6011985140563103	2018-01-30 02:03:54	350	36587	1.4569845000000002E7

only showing top 20 rows

19) Now, summon our good friend happybase to load this dataframe into NoSQL database i.e. Hbase.

20) Create a connection to Hbase, Check if table you want to create already exists and create one if it doesn't exist.

21) Batch load data from dataframe to table created.

```
In [78]: #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"
#get the pointer to a table
def get_table(name):
    open_connection()
    table = connection.table(name)
    close_connection()
    return table
```

```
In [79]: create_table('look_up_table', {'info' : dict(max_versions=5) })
```

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

```
In [85]: #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

    #Creating a rowkey for better data query. RowKey is the cardId .
    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()
```

```
In [86]: batch_insert_data(look_up_table,'look_up_table')
```

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

- 22) Open Putty, login as root user. Go to Hbase Shell and list existing tables.
- 23) Our look\_up\_table should already be appearing there, scan it to see if data is loaded as expected.

```
hbase(main):001:0> list
TABLE
card_transactions
employee
look_up_table
3 row(s) in 0.3340 seconds

=> ["card_transactions", "employee", "look_up_table"]
hbase(main):002:0> █
```

Give command scan 'look\_up\_table' to see data inserted into table.

```
523145603633304      column=info:transaction_date, timestamp=1607880087970, value=2018-01-22 00:56:57
5232083808576685     column=info:UCL, timestamp=1607880086427, value=14120434.4
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
5254025009868430     column=info:card_id, timestamp=1607880087698, value=5254025009868430
5254025009868430     column=info:postcode, timestamp=1607880087698, value=12973
```

Activate V  
Go to Setting



root@ip-10-10-10-10:~

```

6591175617713393      column=info:transaction_date, timestamp=1607880087142, value=2018-01-31 13:10:37
6592184145413632      column=info:UCL, timestamp=1607880086730, value=13734342.65
6592184145413632      column=info:card_id, timestamp=1607880086730, value=6592184145413632
6592184145413632      column=info:postcode, timestamp=1607880086730, value=53186
6592184145413632      column=info:score, timestamp=1607880086730, value=456
6592184145413632      column=info:transaction_date, timestamp=1607880086730, value=2018-01-28 00:54:30
6594248319343442      column=info:UCL, timestamp=1607880086800, value=15065362.77
6594248319343442      column=info:card_id, timestamp=1607880086800, value=6594248319343442
6594248319343442      column=info:postcode, timestamp=1607880086800, value=24927
6594248319343442      column=info:score, timestamp=1607880086800, value=350
6594248319343442      column=info:transaction_date, timestamp=1607880086800, value=2018-01-31 23:42:38
6595638658736751      column=info:UCL, timestamp=1607880087351, value=14005069.97
6595638658736751      column=info:card_id, timestamp=1607880087351, value=6595638658736751
6595638658736751      column=info:postcode, timestamp=1607880087351, value=68328
6595638658736751      column=info:score, timestamp=1607880087351, value=310
6595638658736751      column=info:transaction_date, timestamp=1607880087351, value=2018-01-30 10:50:34
6595814135833988      column=info:UCL, timestamp=1607880087066, value=14332708.84
6595814135833988      column=info:card_id, timestamp=1607880087066, value=6595814135833988
6595814135833988      column=info:postcode, timestamp=1607880087066, value=22508
6595814135833988      column=info:score, timestamp=1607880087066, value=210
6595814135833988      column=info:transaction_date, timestamp=1607880087066, value=2018-01-30 02:03:54
6595928469079750      column=info:UCL, timestamp=1607880087956, value=11824730.01
6595928469079750      column=info:card_id, timestamp=1607880087956, value=6595928469079750
6595928469079750      column=info:postcode, timestamp=1607880087956, value=98349
6595928469079750      column=info:score, timestamp=1607880087956, value=412
6595928469079750      column=info:transaction_date, timestamp=1607880087956, value=2018-01-24 12:38:22
6597703848279563      column=info:UCL, timestamp=1607880087391, value=15250624.49
6597703848279563      column=info:card_id, timestamp=1607880087391, value=6597703848279563
6597703848279563      column=info:postcode, timestamp=1607880087391, value=95699
6597703848279563      column=info:score, timestamp=1607880087391, value=218
6597703848279563      column=info:transaction_date, timestamp=1607880087391, value=2018-01-27 10:51:49
6598830758632447      column=info:UCL, timestamp=1607880087564, value=12685782.48
6598830758632447      column=info:card_id, timestamp=1607880087564, value=6598830758632447
6598830758632447      column=info:postcode, timestamp=1607880087564, value=19421
6598830758632447      column=info:score, timestamp=1607880087564, value=293
6598830758632447      column=info:transaction_date, timestamp=1607880087564, value=2018-01-30 00:18:34
6599900931314251      column=info:UCL, timestamp=1607880087928, value=12487392.07
6599900931314251      column=info:card_id, timestamp=1607880087928, value=6599900931314251
6599900931314251      column=info:postcode, timestamp=1607880087928, value=97423
6599900931314251      column=info:score, timestamp=1607880087928, value=297
6599900931314251      column=info:transaction_date, timestamp=1607880087928, value=2018-01-31 11:25:16
999 row(s) in 2.5910 seconds

```

Activate V  
Go to Setting