

Scripts Execution Screenshots of the execution of the scripts

written This document begins its explanation after loading data from RDS & CSV. Here I'll explain about logic that does relevant analysis as per the rules and feeds the data in the look-up table.

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 |
| 009136568025042 | 308 |
| 009190444424572 | 559 |
| 009250698176266 | 233 |
| 009873334520465 | 298 |
| 011716573646690 | 249 |
| 011877954983420 | 497 |
| 012390918683920 | 407 |
| 012731668664932 | 612 |
+-----+-----+
only showing top 20 rows
```

Card_member table:

```
+-----+-----+-----+-----+-----+-----+
| 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
```

```
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

At first, join CARD_MEMBER & MEMBER_SCORE tables to extract and store credit score of each member.

```
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)
```

```
In [33]: score = score.select('mem_id', 'score', 'card_id')
```

Extract required fields from merged dataset i.e. member ID, credit score and card_id. Next, join both history transaction CSV with score DF which is a merged and extracted data frame from both RDS tables. To calculate the latest transaction date of that card, group the merged dataset on CARD_ID and identify max of transaction date. Write max(transaction_date) to a new column.

```
In [40]: hist = tranf.join(score, tranf.member_id == score.mem_id, how='outer')
```

```
In [41]: hist.count()
```

```
Out[41]: 53210
```

```
In [43]: hist = hist.select('card_id', 'amount', 'postcode', 'pos_id', 'transaction_dt', 'status', 'score')
```

```
In [44]: 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
340379737226464	7439113	91763	315067016872305	03-04-2017 11:43:59	GENUINE	229
340379737226464	8217180	16063	208378790148728	03-05-2017 16:47:43	GENUINE	229
340379737226464	8505852	64070	695556848392133	03-06-2017 03:07:27	GENUINE	229
340379737226464	8535431	29817	683602833507395	04-08-2016 20:59:31	GENUINE	229
340379737226464	6317993	28425	258522244165233	05-05-2017 00:23:45	GENUINE	229
340379737226464	3256860	16845	933410474855991	05-10-2017 15:09:09	GENUINE	229
340379737226464	1423779	97640	789378980336517	06-02-2017 02:10:00	GENUINE	229
340379737226464	3783517	70552	963177679534627	06-12-2016 03:10:30	GENUINE	229
340379737226464	3300714	75750	072728631441941	07-01-2017 05:52:58	GENUINE	229

```
In [53]: look_up_table = history.groupBy('card_id').agg(f.max("transaction_date").alias('transaction_date'))
```

```
In [54]: 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

Join previous last step data frame (score) with look_up_table dataset created above. This step frames all required cols for look_up_table except the UCL.

```
In [57]: look_up_table = look_up_table.join(score, look_up_table.card_id == score.cardid, how='INNER')
```

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
	371885417506954	2018-01-28 14:57:11	741464	19468	551815269280261	GENUINE	599
	5316831626197194	2018-01-29 13:46:32	6716709	40488	704363694703346	GENUINE	227
	6011027251671860	2018-01-28 11:26:31	810486	43437	617450656798765	GENUINE	351

Calculating UCL: Its given in our module that $UCL = \text{Moving Average} + 3 * (\text{Standard Deviation})$

We will first calculate moving average of card amounts for last 10 transactions. For this, as a first step, we create a window over which we group dataframe on card_id such that transactions on same card_id collate and then order them on transaction-date. Which means we figure out all card transactions grouped by card on ask order. Rank each of these rows from 1 being latest and 2 being next latest. Choose only rows whose rank is less than 10, thus only taking top 10 transactions on each card_id. Import

```
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
	345406224887566	1242240	14510	536497882467098	GENUINE	349	2017-06-11 16:31:45	5
	345406224887566	9222549	68358	875905403447795	GENUINE	349	2017-06-10 21:13:03	6
	345406224887566	8726784	64487	617331009748827	GENUINE	349	2017-03-16 03:04:40	7
	345406224887566	2415599	99137	751829480922658	GENUINE	349	2017-03-08 12:29:44	8
	345406224887566	9671941	65614	607206139883123	GENUINE	349	2017-01-21 08:42:47	9

Apply formula of UCL i.e. moving average + 3 * (standard deviation) on above derivations and your UCL should be ready.


```
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

only showing top 20 rows

```
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

only showing top 20 rows

Join the latest dataframe with previous dataframe where you had all data with 'card_id', 'transaction_date', 'score', 'postcode'

```
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

Drop duplicates on this DF to remove redundant transactions done of card_id, transaction date, score & post code. Loading Dataframe to look up table: We take help of our good friend happybase API to perform this task for us. bulk data into Hbase tables. Process involved in creating & loading data into tables:

- 1) Creating connection with hbase
- 2) Checking if table already exists
- 3) Create table as desired if table doesn't already exist.
- 4) Batch insert data into table created in step 3 from final dataframe created above.

```
In [74]: look_up_table = look_up_table.dropDuplicates(['card_id','transaction_date','postcode'])
```

```
In [75]: look_up_table.count()
```

```
Out[75]: 1000
```

```
In [76]: import happybase
#create connection
connection = happybase.Connection('localhost', port=9090 ,autoconnect=False)
```

```
In [77]: def open_connection():
    connection.open()
#close the opened connection
def close_connection():
    connection.close()
#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
```

```
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
```

```
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
```

Once execution is complete, login to putty as root and enter Hbase shell Give command 'list' to see existing tables. Scan 'look_up_table' to see content inside look up table created in pyspark file.

```
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>
```

root@ip-10-0-0-~

```
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
```