Applied full Join on categoryid - it has huge data to handle

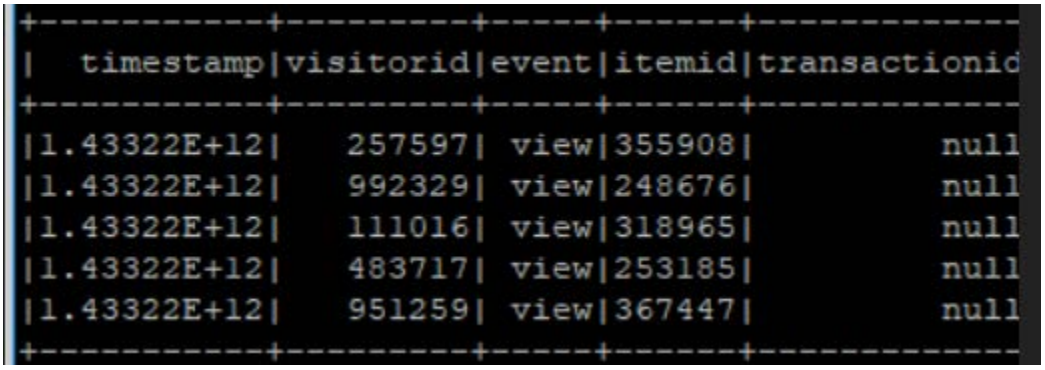
 # Read 'event' data

>>>event = spark.read.options(header=True, inferSchema=True).csv("/user/capstone/event.csv")

>>>event.count()

2756101

>>>event.show(5)



```
+-------------+---------+-----+------+-------------
|    timestamp|visitorid|event|itemid|transactionid
+-------------+---------+-----+------+-------------
|1.43322E+12|   257597| view|355908|         null
|1.43322E+12|   992329| view|248676|         null
|1.43322E+12|   111016| view|318965|         null
|1.43322E+12|   483717| view|253185|         null
|1.43322E+12|   951259| view|367447|         null
+-------------+---------+-----+------+-------------
```

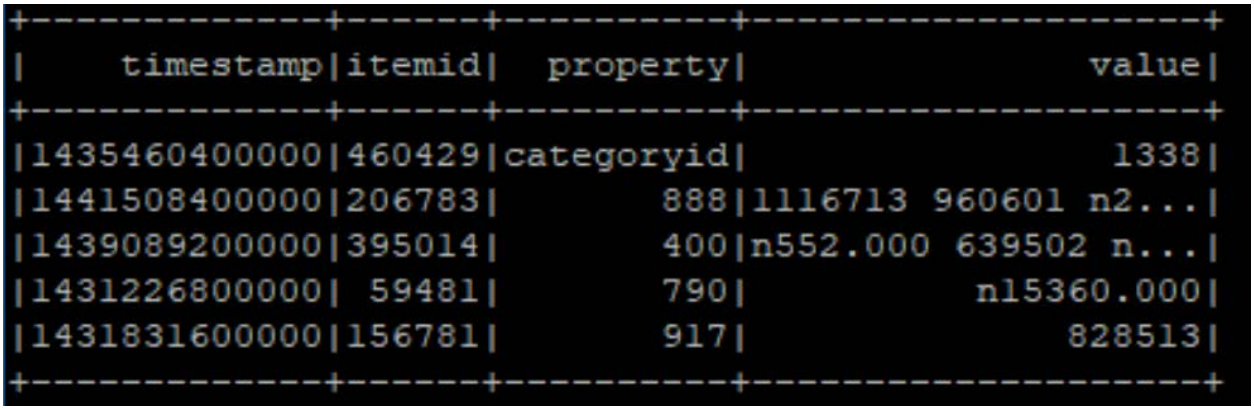
 # Read item_properties_part1 data

>>>item_properties_part1 = spark.read.options(header=True, inferSchema=True).csv("/user/capstone/item_properties_part1.csv")

>>>item_properties_part1.count()

10999999

>>>item_properties_part1.show(5)



```
+-------------+------+----------+--------------------+
|    timestamp|itemid|  property|               value|
+-------------+------+----------+--------------------+
|1435460400000|460429|categoryid|                1338|
|1441508400000|206783|       888|1116713 960601 n2...|
|1439089200000|395014|       400|n552.000 639502 n...|
|1431226800000| 59481|       790|          n15360.000|
|1431831600000|156781|       917|              828513|
+-------------+------+----------+--------------------+
```

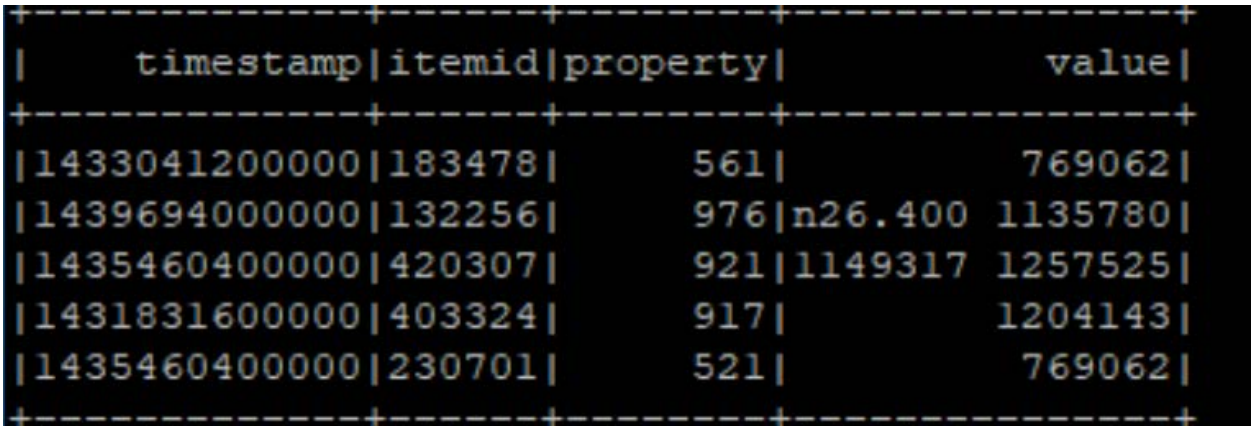
 # Read item_properties_part2 data

>>>item_properties_part2 = spark.read.options(header=True, inferSchema=True).csv("/user/capstone/item_properties_part2.csv")

>>>item_properties_part2.count()

9275903

>>>item_properties_part2.show(5)



timestamp	itemid	property	value
1433041200000	183478	561	769062
1439694000000	132256	976	n26.400 1135780
1435460400000	420307	921	1149317 1257525
1431831600000	403324	917	1204143
1435460400000	230701	521	769062

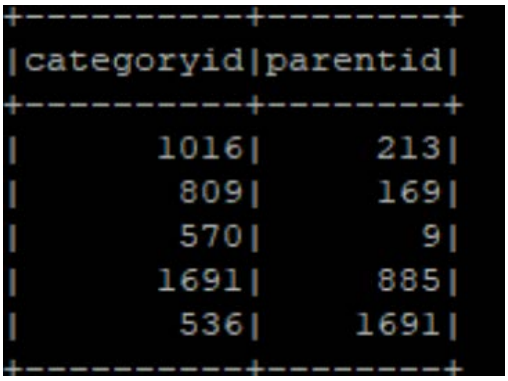
#Read Category_tree data

>>>Category_tree = spark.read.options(header=True, inferSchema=True).csv("/user/capstone/category_tree.csv")

>>>category_tree.count()

1669

>>>category_tree.show(5)



categoryid	parentid
1016	213
809	169
570	9
1691	885
536	1691

 # describe data types

>>>event.describe()

>>>item_properties_part1.describe()

>>>item_properties_part2.describe()

>>>category_tree.describe()

```
>>> event.describe()
DataFrame[summary: string, timestamp: string, visitorid: string, event: string, itemid: string, transactionid: s
tring]
>>> item_properties_part1.describe()
DataFrame[summary: string, timestamp: string, itemid: string, property: string, value: string]
>>> item_properties_part2.describe()
DataFrame[summary: string, timestamp: string, itemid: string, property: string, value: string]
>>> category_tree.describe()
DataFrame[summary: string, categoryid: string, parentid: string]
```

 # Union item_properties_part1 and item_properties_part2

>>>item_properties = item_properties_part1.unionAll(item_properties_part2)

>>>item_properties.count()

20275902

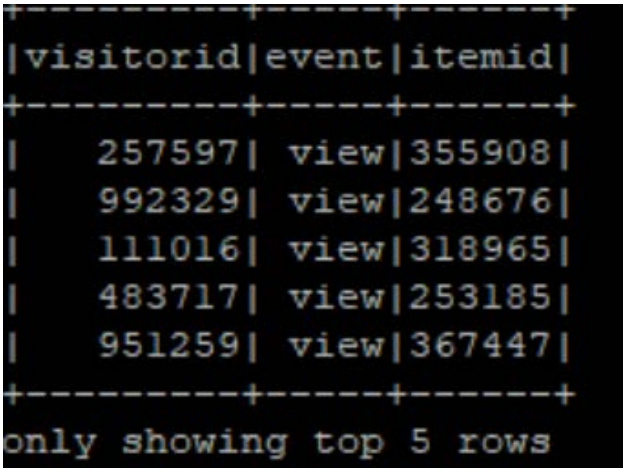
>>>item_properties.show(20)

timestamp	itemid	property	value
1435460400000	460429	categoryid	1338
1441508400000	206783	888	1116713 960601 n2...
1439089200000	395014	400	n552.000 639502 n...
1431226800000	59481	790	n15360.000
1431831600000	156781	917	828513
1436065200000	285026	available	0
1434250800000	89534	213	1121373
1431831600000	264312	6	319724
1433646000000	229370	202	1330310
1434250800000	98113	451	1141052 n48.000

 # Event data cleaning – Remove ‘timestamp’ and ‘transactionid’ # ‘transactionid’ is already included in ‘event’ column

>>>Event_initial_clean = event.drop(‘timestamp’, ‘transactionid’)

>>>Event_initial_clean.show(5)



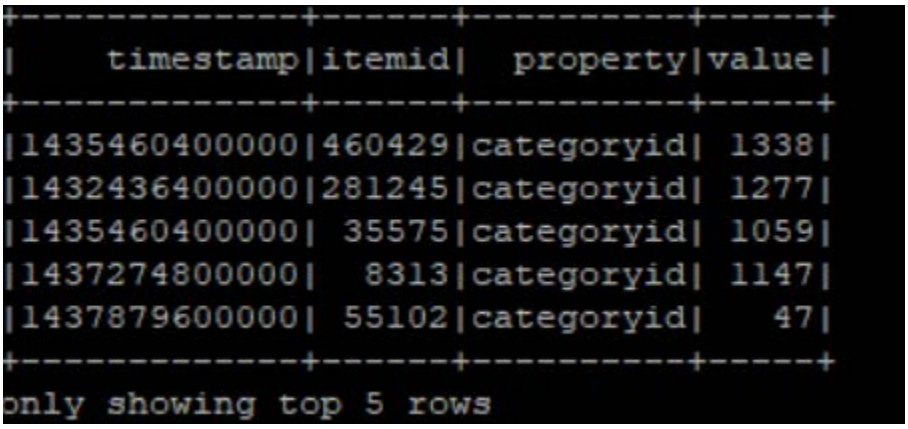
```
+---------+-----+------+
|visitorid|event|itemid|
+---------+-----+------+
|   257597| view|355908|
|   992329| view|248676|
|   111016| view|318965|
|   483717| view|253185|
|   951259| view|367447|
+---------+-----+------+
only showing top 5 rows
```

 # item_properties cleaning

‘value’ column contain categoryid number, availability of product(0 or 1), and prices of the categoryid’s over a period. Keeping the ‘value’ column with availability and prices will mess up the data. Hence, we consider only the ‘value’ with categoryid.

>>>item_properties_clean1 = item_properties[item_properties.property == ‘categoryid’]

>>>item_properties_clean1.show(5)



```
+-------------+------+----------+-----+
|    timestamp|itemid|  property|value|
+-------------+------+----------+-----+
|1435460400000|460429|categoryid| 1338|
|1432436400000|281245|categoryid| 1277|
|1435460400000| 35575|categoryid| 1059|
|1437274800000|  8313|categoryid| 1147|
|1437879600000| 55102|categoryid|   47|
+-------------+------+----------+-----+
only showing top 5 rows
```

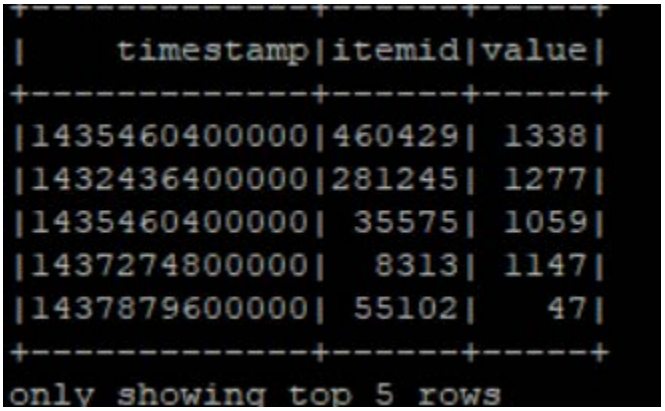
 # ‘property’ and ‘value’ columns are maintaining same data in category (redundant)

>>>item_properties_clean2 = item_properties_clean1.drop(‘property’)

>>>item_properties_clean3.count()

788214

>>>item_properties_clean2.show(5)

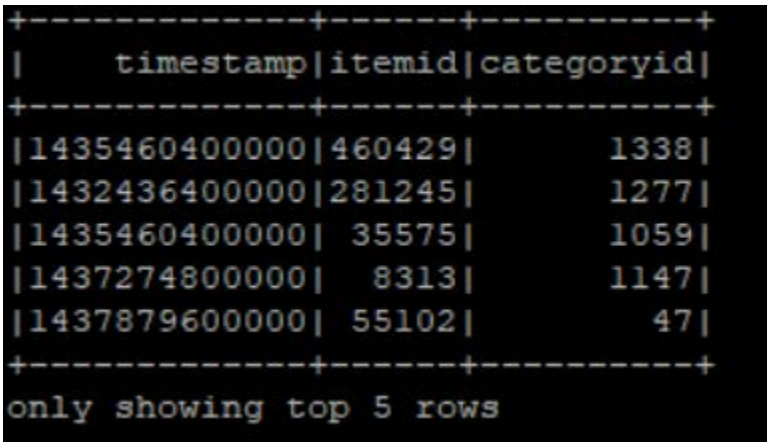


```
+-------------+------+-----+
|    timestamp|itemid|value|
+-------------+------+-----+
|14354604000000|460429| 1338|
|14324364000000|281245| 1277|
|14354604000000| 35575| 1059|
|14372748000000|  8313| 1147|
|14378796000000| 55102|   47|
+-------------+------+-----+
only showing top 5 rows
```

 # Creating dataframe

>>>item_properties_clean3 = item_properties_clean2.toDF(‘timestamp’,’itemid’,’categoryid’)

>>>item_properties_clean3.show(5)



```
+-------------+------+----------+
|    timestamp|itemid|categoryid|
+-------------+------+----------+
|14354604000000|460429|      1338|
|14324364000000|281245|      1277|
|14354604000000| 35575|      1059|
|14372748000000|  8313|      1147|
|14378796000000| 55102|        47|
+-------------+------+----------+
only showing top 5 rows
```

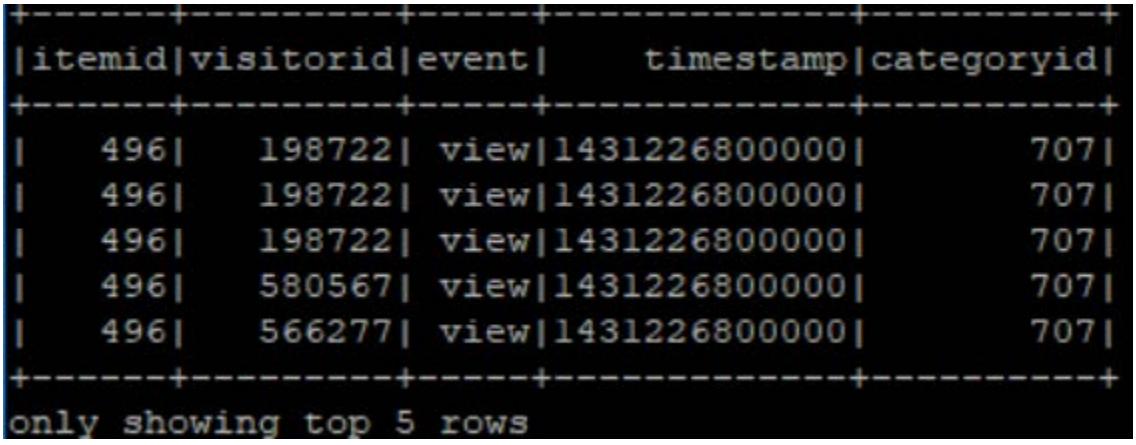
 # Join event and item_properties columns after cleaning

>>>Event_Item = Event_initial_clean.join(item_properties_clean3, [‘itemid’])

>>>Event_Item.count()

2120911

>>>Event_Item.show(5)



```
+------+---------+-----+-------------+----------+
|itemid|visitorid|event|    timestamp|categoryid|
+------+---------+-----+-------------+----------+
|   496|   198722| view|1431226800000|       707|
|   496|   198722| view|1431226800000|       707|
|   496|   198722| view|1431226800000|       707|
|   496|   580567| view|1431226800000|       707|
|   496|   566277| view|1431226800000|       707|
+------+---------+-----+-------------+----------+
only showing top 5 rows
```

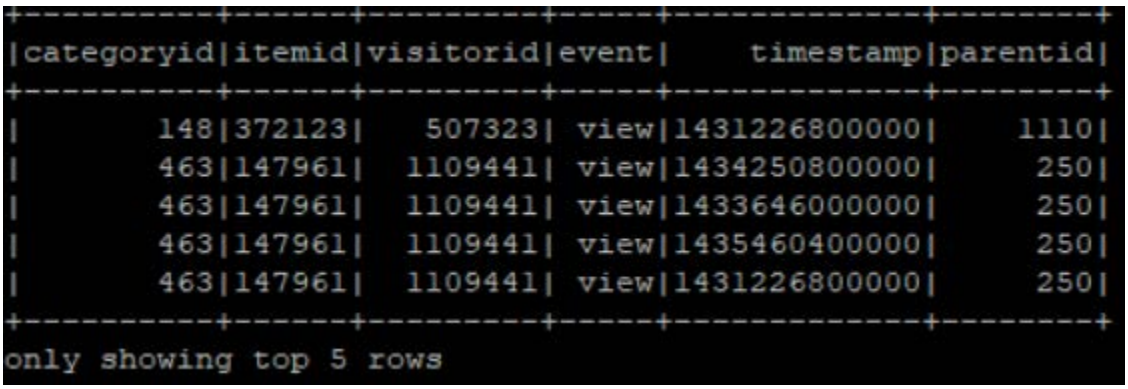
 # Joining Event_itemproperties with category_tree

>>>Event_item_parent = Event_item.join(category_tree, [‘categoryid’])

>>>Event_item_parent.count()

2120911

>>>Event_item_parent.show(5)



```
+----------+------+---------+-----+-------------+--------+
|categoryid|itemid|visitorid|event|    timestamp|parentid|
+----------+------+---------+-----+-------------+--------+
|       148|372123|   507323| view|1431226800000|    1110|
|       463|147961|  1109441| view|1434250800000|     250|
|       463|147961|  1109441| view|1433646000000|     250|
|       463|147961|  1109441| view|1435460400000|     250|
|       463|147961|  1109441| view|1431226800000|     250|
+----------+------+---------+-----+-------------+--------+
only showing top 5 rows
```

 # Call saved file in hdfs

>>> file = spark.read.options(header=True, inferSchema=True).csv("/user/capstone/Event_item_parent")

>>>file.count()

2120911

>>>file.show(5)

```
+----------+------+---------+-----+-------------+--------+
|categoryid|itemid|visitorid|event|    timestamp|parentid|
+----------+------+---------+-----+-------------+--------+
|       707|   496|   198722| view|1431226800000|     561|
|       707|   496|   198722| view|1431226800000|     561|
|       707|   496|   198722| view|1431226800000|     561|
|       707|   496|   580567| view|1431226800000|     561|
|       707|   496|   566277| view|1431226800000|     561|
+----------+------+---------+-----+-------------+--------+
only showing top 5 rows
```

 # Assign the label for 'event' column

>>> data_A = df.replace({'view':'1'})

>>>data_B = data_A.replace({'addtocart':'2'})

>>>data = data_B.replace({'transaction':'3'})

>>>data.count()

2120911

>>>data.show(5)

```
+----------+------+--------+-----+-------------+--------+
|categoryid|itemid|visitoid|event|    timestamp|parentid|
+----------+------+--------+-----+-------------+--------+
|       707|   496|  198722|    1|1431226800000|     561|
|       707|   496|  198722|    1|1431226800000|     561|
|       707|   496|  198722|    1|1431226800000|     561|
|       707|   496|  580567|    1|1431226800000|     561|
|       707|   496|  566277|    1|1431226800000|     561|
+----------+------+--------+-----+-------------+--------+
only showing top 5 rows
```

 # extracts the columns to apply recommender engine

>>>df_mat= data[‘visitorid’,’itemid’,’event’]

>>>df_mat.show(5)

```
+--------+------+-----+
|visitoid|itemid|event|
+--------+------+-----+
|  198722|   496|    1|
|  198722|   496|    1|
|  198722|   496|    1|
|  580567|   496|    1|
|  566277|   496|    1|
+--------+------+-----+
```

#Model:

a) Apply Recomm on value[visitorid, itemid,event]

```
>>> df_mat.coalesce(1).write.option('header','true').csv('/user/capstone/df_mat.
csv')
>>> df_mat = sc.textFile('/user/capstone/df_mat')
>>> type(df_mat)
<class 'pyspark.rdd.RDD'>
```

b) Getting saved file from HDFS and remove ‘header’

```
>>> df_mat = sc.textFile('/user/capstone/df_mat.csv')
>>> df_mat.first()
u'visitoid,itemid,event'
>>> df_mat.count()
2120912
>>> header = df_mat.first()
>>> data_no__header = df_mat.filter(lambda x: x!=header)
>>> data_no__header.count()
2120911
>>> data_no__header.first()
u'198722,496,1'
```

c) Getting saved no header file from HDFS and and build the recommendation model applying ALS (Alternative least squares)

```
>>> data = sc.textFile('/user/capstone/data_no_header')
>>> data.take(1)
[u'198722,496,1']
>>> ratings = data.map(lambda x: x.split(','))\
... .map(lambda x: Rating(int(x[0]), int(x[1]), float(x[2])))
```

```
>>> from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rati
ng
```

```
>>> rank = 10
>>> numIterations = 10
>>> model = ALS.train(ratings, rank, numIterations)
```

```
>>> model
<pyspark.mllib.recommendation.MatrixFactorizationModel object at 0x2386d50>
```

d)Evaluate the model on training data

```
>>> testdata = ratings.map(lambda x: (x[0], x[1]))
>>> testdata.take(1)
[(198722, 496)]
>>> testdata.take(5)
[(198722, 496), (198722, 496), (198722, 496), (580567, 496), (566277, 496)]
>>> predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]),r[2]))
>>> predictions.take(5)
[Stage 199:=============================================>             (3 + 1) / 4]
18/06/21 00:25:47 WARN Executor: Managed memory leak detected; size = 78748626 b
ytes, TID = 148
[((1080592, 234255), 0.99215995872064688), ((1022816, 321619), 0.996172664938656
19), ((637100, 74296), 0.99920660553012408), ((637100, 74296), 0.999206605530124
08), ((637100, 192463), 0.98235062487273606)]
>>> predictions.top(5)
[((1407579, 2521), 0.99599431280887474), ((1407576, 356208), 0.99741931266060202
), ((1407575, 121220), 0.99602039223281402), ((1407573, 82278), 0.99378353003843
323), ((1407573, 82278), 0.99378353003843323)]
```

```
>>> top_20 = prediction.top(20)
```

```
>>> top_20
[((1407579, 2521), 0.99599431280887474), ((1407576, 356208), 0.99741931266060202
), ((1407575, 121220), 0.99602039223281402), ((1407573, 82278), 0.99378353003843
323), ((1407573, 82278), 0.99378353003843323), ((1407567, 219086), 1.00466745940
84897), ((1407567, 183650), 0.99425482313910973), ((1407567, 183650), 0.99425482
313910973), ((1407567, 183650), 0.99425482313910973), ((1407567, 183650), 0.9942
5482313910973), ((1407567, 183650), 0.99425482313910973), ((1407567, 183650), 0.
99425482313910973), ((1407567, 183650), 0.99425482313910973), ((1407567, 183650)
, 0.99425482313910973), ((1407567, 183650), 0.99425482313910973), ((1407567, 183
650), 0.99425482313910973), ((1407567, 183650), 0.99425482313910973), ((1407567,
 183650), 0.99425482313910973), ((1407567, 183650), 0.99425482313910973), ((1407
567, 183650), 0.99425482313910973)]
```

```
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error = " + str(MSE))
```

```
>>> ratesAndPreds.top(20)
[((1407579, 2521), (1.0, 0.99599431280887474)), ((1407576, 356208), (1.0, 0.99741931
266060202)), ((1407575, 121220), (1.0, 0.99602039223281402)), ((1407573, 82278), (1.
0, 0.99378353003843323)), ((1407573, 82278), (1.0, 0.99378353003843323)), ((1407573,
 82278), (1.0, 0.99378353003843323)), ((1407573, 82278), (1.0, 0.99378353003843323))
, ((1407567, 219086), (1.0, 1.0046674594084897)), ((1407567, 183650), (1.0, 0.994254
82313910973)), ((1407567, 183650), (1.0, 0.9942548231391097 3)), ((1407567, 183650),
(1.0, 0.99425482313910973)), ((1407567, 183650), (1.0, 0.99425482313910973)), ((1407
567, 183650), (1.0, 0.99425482313910973)), ((1407567, 183650), (1.0, 0.9942548231391
0973)), ((1407567, 183650), (1.0, 0.99425482313910973)), ((1407567, 183650), (1.0, 0
.99425482313910973)), ((1407567, 183650), (1.0, 0.99425482313910973)), ((1407567, 18
3650), (1.0, 0.99425482313910973)), ((1407567, 183650), (1.0, 0.99425482313910973)),
 ((1407567, 183650), (1.0, 0.99425482313910973))]
```

```
>>> first_user = model.userFeatures().take(1)[0]
>>> first_user
(52, array('d', [-0.26322278380393982, 0.1056523397564888, -0.12677446007728577, -0.
0034060413017868996, -0.11442477256059647, 0.19780953228473663, 0.30431264638900757,
 -0.28618881106376648, 0.2259412556886673, -0.25969997048377991]))
>>>
```

```
>>> first_product = model.productFeatures().take(1)[0]
>>> first_product
(4, array('d', [-0.13493116199970245, 0.65108019113540649, -0.27421149611473083, 0.1
3661985099315643, -0.012780096381902695, 0.1139710545539856, -0.074562638998031616,
-0.26817235350608826, -0.59668350219726562, 0.024816812947392464]))
>>> latents = first_product[1]
>>> latents
array('d', [-0.13493116199970245, 0.65108019113540649, -0.27421149611473083, 0.13661
985099315643, -0.012780096381902695, 0.1139710545539856, -0.074562638998031616, -0.2
6817235350608826, -0.59668350219726562, 0.024816812947392464])
>>> len(latents)
10
>>>
```

Collaborating filtering on 'event' data by using left outer join with item categoryid only- predent data is explicit

i) Load event and removing transactionid because it has more NAs and redundant also

```
>>> eve = spark.read.options(header=True,inferSchema=True).csv('/user/capstone/events.csv')
>>> eve_1 = eve.drop('transactionid')
>>> eve_1.show(5)
+-------------+---------+-----+------+
|    timestamp|visitorid|event|itemid|
+-------------+---------+-----+------+
|1433221332117|   257597| view|355908|
|1433224214164|   992329| view|248676|
|1433221999827|   111016| view|318965|
|1433221955914|   483717| view|253185|
|1433221337106|   951259| view|367447|
+-------------+---------+-----+------+
only showing top 5 rows

>>> eve_1.count()
2756101
part1')m1 = spark.read.option(header=True, inferSchema=True).csv('/user/capstone/item_properties
```

ii) Merging item_part1 and item_part2 and extracting 'categoryid's to make model size less

```
>>> item1 = spark.read.options(header=True, inferSchema=True).csv('/user/capstone/item_properties
_part1.csv')                        .count()
[Stage 9:>                                                          (0 + 4) / 4]
_part1')spark.read.options(header=True, inferSchema=True).csv('/user/capstone/item_properties_

10999999
>>>
>>>
>>>
>>> item2 = spark.read.options(header=True, inferSchema=True).csv('/user/capstone/item_properties_part2.csv')
>>> item2.count()
9275903
>>> item = item1.unionAll(item2)
>>> item.count()
20275902
>>> item_only_cat = item[item.propert == 'categoryid']
>>> item_only_cat = item[item.property == 'categoryid']
>>> item_only_cat.count()
788214
>>> event_1.coalesce(1).write.option('header','true').csv('/user/capstone/eve_4col.csv')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'event_1' is not defined
>>> eve_1.coalesce(1).write.option('header','true').csv('/user/capstone/eve_4col.csv')
>>> itema = item.drop('timestamp')
>>> itema.show(5)
+------+----------+--------------------+
|itemid|  property|               value|
+------+----------+--------------------+
|460429|categoryid|                1338|
|206783|       888|1116713 960601 n2...|
|395014|       400|n552.000 639502 n...|
| 59481|       790|          n15360.000|
|156781|       917|              828513|
+------+----------+--------------------+
only showing top 5 rows
```

iii) Joining the event, item and category_tree (considered only left join with itemid and categoryid)

```
>>> event_item = eve_1.join(itema,'itemid','left')
>>> event_item.count()
170797695
>>> event_item = eve_1.join(itema,'itemid','right')
>>> event_item_r = eve_1.join(itema,'itemid','right')
>>> event_item_r.count()
180637859
>>> event_item_i = eve_1.join(itema,'itemid','inner')
>>> event_item_i.count()
170542110
>>> event_itemoncat = eve_1.join(item_only_cat,'itemid','left')
>>> event_itemoncat.count()
5900263
>>> event_itemoncat.show(10)
+------+-------------+---------+-----+-------------+----------+-----+
|itemid|    timestamp|visitorid|event|    timestamp|  property|value|
+------+-------------+---------+-----+-------------+----------+-----+
|   463|1434837366894|   239457| view|         null|      null| null|
|   463|1435019299963|   838754| view|         null|      null| null|
|   463|1435157394016|  1407404| view|         null|      null| null|
|   496|1433283482363|   198722| view|1431226800000|categoryid|  707|
|   496|1433284181852|   198722| view|1431226800000|categoryid|  707|
|   496|1433282355029|   198722| view|1431226800000|categoryid|  707|
|   496|1433228455478|   580567| view|1431226800000|categoryid|  707|
|   496|1433345947990|   566277| view|1431226800000|categoryid|  707|
|   496|1433352568025|  1133742| view|1431226800000|categoryid|  707|
|   496|1433374113815|   198722| view|1431226800000|categoryid|  707|
+------+-------------+---------+-----+-------------+----------+-----+
only showing top 10 rows
```

iv) Rename the 'value' variable as categoryid and drop the property variable which contains only categoryid text.

```
>>> cat = spark.read.options(header=True, inferSchema=True).csv('/user/capstone/category_tree.csv')
>>> event_itemoncat_dp = event_itemoncat.drop('timestamp','property')
>>> event_itemoncat_dp.show(5)
+------+---------+-----+-----+
|itemid|visitorid|event|value|
+------+---------+-----+-----+
|   463|   239457| view| null|
|   463|   838754| view| null|
|   463|  1407404| view| null|
|   496|   198722| view|  707|
|   496|   198722| view|  707|
+------+---------+-----+-----+
only showing top 5 rows

>>> rename_eve_item = event_itemoncat_dp.toDF('itemid','visitorid','event','categoryid')
>>> rename_eve_item.show(5)
+------+---------+-----+----------+
|itemid|visitorid|event|categoryid|
+------+---------+-----+----------+
|   463|   239457| view|      null|
|   463|   838754| view|      null|
|   463|  1407404| view|      null|
|   496|   198722| view|       707|
|   496|   198722| view|       707|
+------+---------+-----+----------+
only showing top 5 rows
```

```
>>> event_item_cat = rename_eve_itme.join(cat, 'categoryid','left')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'rename_eve_itme' is not defined
>>> event_item_cat = rename_eve_item.join(cat, 'categoryid','left')
>>> event_item_cat.count()
5900263
>>> event_item_cat.show(5)
+----------+------+---------+-----+--------+
|categoryid|itemid|visitorid|event|parentid|
+----------+------+---------+-----+--------+
|      null|   463|   239457| view|    null|
|      null|   463|   838754| view|    null|
|      null|   463|  1407404| view|    null|
|       707|   496|   198722| view|     561|
|       707|   496|   198722| view|     561|
+----------+------+---------+-----+--------+
only showing top 5 rows

>>> event_item_cat.coalesce(1).write.option('header','true').csv('/user/capstone/eve_item_catid.csv')
```

v) giving labels to the event values like ‘view = 1’, ‘addtocart = 2’, ‘transaction = 3’

```
>>> # giving labels
...
>>> R1 = event_item_cat.replace({'view':'1'})
>>> R2 = R1.replace({'addtocart':'2'})
>>> R3 = R2.replace({'transaction':'3'})
>>> R3.coalesce(1).write.option('header','true').csv('/user/capstone/R.csv')
>>> R = sc.textFile('/user/capstone/R.csv')
>>> R.first()
u'categoryid,itemid,visitorid,event,parentid'
```

vi) Extracting the visitorid , itemid, and event to apply ALS

```
>>> extract_eve = R3['visitorid','itemid','event']
>>> extract_eve.show(5)
+---------+------+-----+
|visitorid|itemid|event|
+---------+------+-----+
|   239457|   463|    1|
|   838754|   463|    1|
|  1407404|   463|    1|
|   198722|   496|    1|
|   198722|   496|    1|
+---------+------+-----+
only showing top 5 rows

>>> extract_eve.count()
5900263
>>> extract_eve.coalesce(1).write.option('header','true').csv('/user/capstone/fina
l_eve_extract.csv')
```

vii) removing header to make a model matrix

```
>>> Re = sc.textFile('/user/capstone/final_eve_extract.csv')
>>> Re.first()
u'visitorid,itemid,event'
>>> header = Re.filter(lambda x: x!=header)
>>> data = Re.filter(lambda x: x!=header)
>>> ratings = data.map(lambda l: l.split(','))\
... .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
>>> rank = 10
>>> numIterations = 10
```

```
>>> ratings.take(5)
[Rating(user=239457, product=463, rating=1.0), Rating(user=838754, product=463, rating=
1.0), Rating(user=1407404, product=463, rating=1.0), Rating(user=198722, product=496, r
ating=1.0), Rating(user=198722, product=496, rating=1.0)]
```

viii) Model for explicit data and finding MSE

```
>>> rank = 10
>>> numIterations = 10
>>> model = ALS.train(ratings, rank, numIterations)
18/07/05 13:35:05 WARN BLAS: Failed to load implementation from: com.github.fommil.netl
ib.NativeSystemBLAS
18/07/05 13:35:05 WARN BLAS: Failed to load implementation from: com.github.fommil.netl
ib.NativeRefBLAS
[Stage 137:>                                                        (0 + 4) / 4]18/07/0
5 13:35:11 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.Na
tiveSystemLAPACK
18/07/05 13:35:11 WARN LAPACK: Failed to load implementation from: com.github.fommil.ne
tlib.NativeRefLAPACK
>>> testdata = ratings.map(lambda p: (p[0], p[1]))
>>> predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
>>> df = sqlContext.createDataFrame(ratings)
>>> df.select('user').dist()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/hdp/current/spark2-client/python/pyspark/sql/dataframe.py", line 1020, in
__getattr__
    "'%s' object has no attribute '%s'" % (self.__class__.__name__, name))
AttributeError: 'DataFrame' object has no attribute 'dist'
>>> df.select('user').distinct()
DataFrame[user: bigint]
>>> ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
>>> MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
>>> print("Mean Squared Error = " + str(MSE))
Mean Squared Error = 0.0539044229875
```

ix) Save and load the model

```
>>> model.save(sc, '/user/capstone/CollaborativeFilter')
>>> sameModel = MatrixFactorizationModel.load(sc, '/user/capstone/CollaborativeFilt
18/07/05 14:50:38 WARN MatrixFactorizationModel: User factor does not have a partit
r. Prediction on individual records could be slow.
18/07/05 14:50:38 WARN MatrixFactorizationModel: User factor is not cached. Predict
could be slow.
18/07/05 14:50:38 WARN MatrixFactorizationModel: Product factor does not have a par
oner. Prediction on individual records could be slow.
18/07/05 14:50:38 WARN MatrixFactorizationModel: Product factor is not cached. Pred
on could be slow.
18/07/05 14:50:38 WARN MatrixFactorizationModelWrapper: User factor does not have a
titioner. Prediction on individual records could be slow.
18/07/05 14:50:38 WARN MatrixFactorizationModelWrapper: User factor is not cached.
iction could be slow.
18/07/05 14:50:38 WARN MatrixFactorizationModelWrapper: Product factor does not hav
partitioner. Prediction on individual records could be slow.
18/07/05 14:50:38 WARN MatrixFactorizationModelWrapper: Product factor is not cache
rediction could be slow.
```

Applying recommendation model on implicit data as our data is implicit

#building model using ALS on implicit data

```
>>> model_implicit = ALS.trainImplicit(ratings, rank, numIterations, alpha=0.01)
>>> ratings = data.map(lambda l: l.split(','))\
...     .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
>>> rank = 10
>>> numIterations = 10
>>> model_implicit = ALS.train(ratings, rank, numIterations)
>>>
>>>
>>> # Evaluate the model on training data
...
>>> testdata = ratings.map(lambda p: (p[0], p[1]))
>>> testdata.take(5)
[(239457, 463), (838754, 463), (1407404, 463), (198722, 496), (198722, 496)]
>>>
```

i) Evaluating the model on training data

```
>>> testdata_1 = ratings.map(lambda p: (p[0], p[1]))
>>> testdata_1.take(5)
[(239457, 463), (838754, 463), (1407404, 463), (198722, 496), (198722, 496)]
```

```
>>> predictions_1 = model_implicit.predictAll(testdata_1).map(lambda r: ((r[0], r[1]),
r[2]))
>>> predictions_1.take(5)
[Stage 1553:=============================================>             (3 + 1) / 4]18/07/0
5 17:30:52 WARN Executor: Managed memory leak detected; size = 168332422 bytes, TID = 3
056
[((185012, 79378), 0.994403363814990 84), ((612052, 189522), 0.997701033260125 94), ((354
796, 351186), 0.994616702079937 46), ((1022816, 321619), 0.996165235525543 37), ((182316,
 141578), 0.995251232307345 58)]
```

```
>>> ratesAndPreds_1 = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions_1)
>>> ratesAndPreds_1.take(5)
[((100417, 11001), (1.0, 0.9992127044901975)), ((1115684, 62312), (1.0, 0.9954580296170
0547)), ((380733, 418631), (1.0, 0.9526427142255 3026)), ((199705, 218329), (1.0, 0.9992
0319220961762)), ((245669, 28659), (1.0, 0.9956777159227 4756))]
```

```
>>> MSE = ratesAndPreds_1.map(lambda r: ((r[0], r[1]), r[2])).join(predictions_1)
>>> print("Mean Squared Error = " + str(MSE))
Mean Squared Error = PythonRDD[1526] at RDD at PythonRDD.scala:48
>>> MSE = ratesAndPreds_1.map(lambda r: (r[1][0] - r[1][1])**2).mean()
>>> print("Mean Squared Error = " + str(MSE))
Mean Squared Error = 0.0539016794671
>>>
```

distinct user and count

```
>>> df = sqlContext.createDataFrame(ratings)
>>> df.select('user').distinct().show(10)
[Stage 1613:============================>                            (1 + 1) / 2]18/07/0
5 18:19:44 WARN Executor: Managed memory leak detected; size = 4456448 bytes, TID = 307
3
+-------+
|   user|
+-------+
| 324572|
| 604324|
|1196638|
| 457164|
|1037352|
| 416433|
|1221208|
|1034173|
| 271690|
|1341265|
+-------+
only showing top 10 rows
```

```
>>> df.select('user').distinct().count()
1407580
```

Product distinct and count

```
>>> df.select('product').distinct().count()
235061
>>> df.select('product').distinct().show(5)
[Stage 1671:============================>                            (1 + 1) / 2]18/07/0
5 18:33:43 WARN Executor: Managed memory leak detected; size = 4456448 bytes, TID = 349
1
+-------+
|product|
+-------+
|  15846|
|  41988|
| 310728|
| 387799|
|  62526|
+-------+
only showing top 5 rows
```

group By user

```
>>> df.groupBy('user').count().take(10)
[Stage 1673:============================>                            (1 + 1) / 2]18/
5 18:38:14 WARN Executor: Managed memory leak detected; size = 4456448 bytes, TID =
4
[Row(user=324572, count=2), Row(user=604324, count=8), Row(user=1196638, count=29),
(user=457164, count=2), Row(user=1037352, count=1), Row(user=416433, count=5), Row(
=1221208, count=1), Row(user=1034173, count=1), Row(user=271690, count=1), Row(user
1265, count=1)]
>>>
```

group by event

```
>>> df.groupBy('rating')
<pyspark.sql.group.GroupedData object at 0x300fc90>
>>> df.groupBy('rating').count().show()
+------+-------+
|rating|  count|
+------+-------+
|   1.0|5706279|
|   3.0|  44127|
|   2.0| 149857|
+------+-------+
```

Import python libraries

```
>>> import pygtk as plt
>>> import numpy as np
```

Group by user and their statistics

```
>>> df.groupBy('user').count().select('count').describe().show()
+-------+------------------+
|summary|             count|
+-------+------------------+
|  count|           1407580|
|   mean| 4.191778087213516|
| stddev|28.912479033833705|
|    min|                 1|
|    max|             13751|
+-------+------------------+
```

crosstab with user and event

```
>>> df.stat.Crosstab('user','rating').show()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'DataFrameStatFunctions' object has no attribute 'Crosstab'
>>> df.stat.crosstab('user','rating').show()
18/07/05 19:14:21 WARN StatFunctions: The maximum limit of 1e6 pairs have been coll
d, which may not be all of the pairs. Please try reducing the amount of distinct it
in your columns.
+-----------+---+---+---+
|user_rating|1.0|2.0|3.0|
+-----------+---+---+---+
|     669325|  1|  0|  0|
|     834212|  1|  0|  0|
|    1087255|  1|  0|  0|
|     348877|  1|  0|  0|
|    1253013|  8|  0|  0|
|     177494|  1|  0|  0|
|     868559|  1|  0|  0|
|    1192557|  1|  0|  0|
|     864012|  1|  0|  0|
|    1283772| 18|  0|  0|
|     592356|  1|  0|  0|
|    1348443|  1|  0|  0|
|     922480| 21|  0|  0|
|     504671|  1|  0|  0|
|      53767|  1|  0|  0|
|     179367| 18|  0|  0|
|     140403|  1|  0|  0|
|      32571|  1|  0|  0|
|     374171|  1|  0|  0|
|     106805|  1|  0|  0|
+-----------+---+---+---+
only showing top 20 rows
```

User rating with mean

```
>>> df.groupBy('user').agg({'rating':'mean'}).take(5)
[Stage 1700:============================>                          (1 + 1) / 2]18/07/0
5 19:20:01 WARN Executor: Managed memory leak detected; size = 4456448 bytes, TID = 430
6
[Row(user=324572, avg(rating)=1.0), Row(user=604324, avg(rating)=1.0), Row(user=1196638
, avg(rating)=1.4137931034482758), Row(user=457164, avg(rating)=1.0), Row(user=1037352,
 avg(rating)=1.0)]
>>>
```

creating train and test data

```
>>> (training,test) = ratings.randomSplit([0.8,0.2])
>>> training.count()
4718867
>>> test.count()
1181396
```

model creation with train data

```
>>> rank = 10
>>> numIterations = 10
>>> model_A = ALS.trainImplicit(ratings, rank, numIterations, alpha=0.01)
>>> model_A.save(sc, '/user/capstone/model_A)
  File "<stdin>", line 1
    model_A.save(sc, '/user/capstone/model_A)
                                             ^
SyntaxError: EOL while scanning string literal
>>> model_A.save(sc, '/user/capstone/model_A')
>>> model_A
<pyspark.mllib.recommendation.MatrixFactorizationModel object at 0x321dcd0>
>>>
```

creating test data

```
>>> model_A.save(sc, '/user/capstone/model_A')
>>> model_A
<pyspark.mllib.recommendation.MatrixFactorizationModel object at 0x321dcd0>
>>> testdata = test.map(lambda x: (x[0], x[1])
... )
>>> type(testdata)
<class 'pyspark.rdd.PipelinedRDD'>
>>> test.take(5)
[Rating(user=198722, product=496, rating=1.0), Rating(user=566277, product=496, rat
1.0), Rating(user=198722, product=496, rating=1.0), Rating(user=823877, product=496
ting=1.0), Rating(user=857186, product=496, rating=1.0)]
```

predicting the user and itemid

```
>>> pred_ind = model_A.predict(198722,496)
>>> pred_ind
1.6339817636488503e-06
>>> pred_ind = model_A.predict(566277,496)
>>> pred_ind
2.8556375466776656e-07
>>> predictions.saveAsTextFile('/user/capstone/predictions')
[Stage 2430:>                                                                 (0
```

Top N items for recommendations

```
>>> # top N items
...
>>> recommendItemsToUsers = model_A.recommendProductsForUsers(10)
>>> recommendItemsToUsers.count()
1407580
>>> recommendItemsToUsers.take(2)
18/07/06 02:24:08 WARN Executor: Managed memory leak detected; size = 39189274 bytes, T
ID = 4560
[(618672, (Rating(user=618672, product=46156, rating=2.6653154726532398e-06), Rating(us
er=618672, product=316472, rating=2.5774824119438816e-06), Rating(user=618672, product=
291877, rating=2.4457161484510758e-06), Rating(user=618672, product=158666, rating=1.85
20258671449349e-06), Rating(user=618672, product=210087, rating=1.7954638235971775e-06)
, Rating(user=618672, product=390591, rating=1.7247702012026395e-06), Rating(user=61867
2, product=296619, rating=1.6917609335572161e-06), Rating(user=618672, product=332629,
rating=1.5758279834828376e-06), Rating(user=618672, product=190070, rating=1.5583113373
173065e-06), Rating(user=618672, product=56782, rating=1.5391493506341836e-06))), (1080
592, (Rating(user=1080592, product=46156, rating=0.0030035864504229489), Rating(user=10
80592, product=210087, rating=0.0024234410923037993), Rating(user=1080592, product=3201
30, rating=0.0021077717181816023), Rating(user=1080592, product=316472, rating=0.001871
0356741093873), Rating(user=1080592, product=37254, rating=0.0017088186335901519), Rati
ng(user=1080592, product=113535, rating=0.0016372162067537558), Rating(user=1080592, pr
oduct=332629, rating=0.0013604436175994514), Rating(user=1080592, product=56782, rating
=0.0012634008107975556), Rating(user=1080592, product=414460, rating=0.0012033586248367
271), Rating(user=1080592, product=98899, rating=0.0011897670202083941)))]
```

```
>>> recommendItemsToUsers.saveAsTextFile('/user/capstone/recommenderItemToUsers')
```