1. Hadoop Streaming with Python

For all questions in this section, provide

- the python scripts you have written
- the Hadoop streaming command you used to run the job(s) on the Hadoop cluster
- The job/application ID that was automatically assigned to each mapreduce job that you have run for each exercise

Exercise 1 – Counting popularity of movies

a) Write a python mapper and reducer script to count the number of users that have rated each movie (which is an estimate of a movie's popularity). Output should be in the form (itemID \t number of users rated itemID). Define the job name as "job_countByItemID" and run the streaming job.

mapper.py

```
#!/usr/bin/python
import sys
for line in sys.stdin:
    line = line.strip()
    user_id, item_id, rating, timestamp = line.split()
    print '%s\t%s' % (item_id, 1)
```

reducer.py

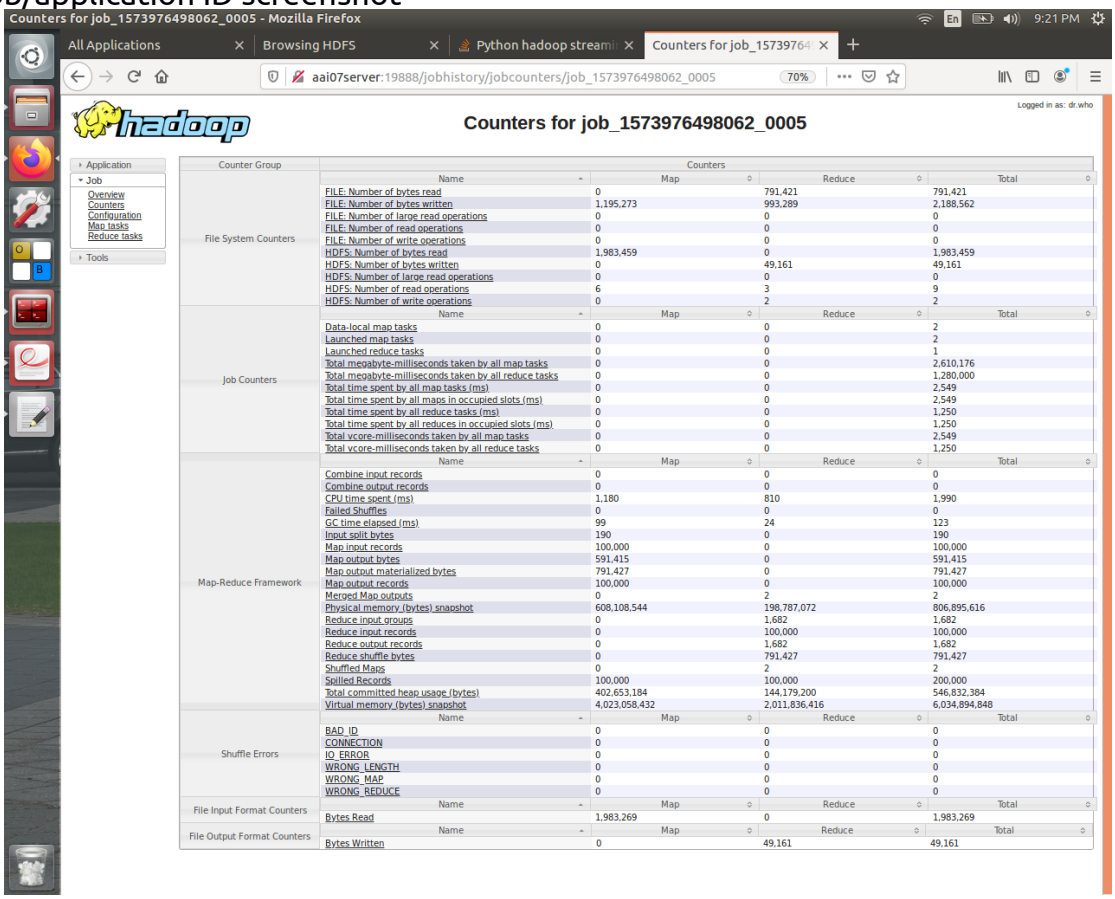
```
#!/usr/bin/python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
                    print '%s\tnumber of users rated\t%s' % (current_count,
current_word)
        current_count = count
        current_word = word
```

```
if current_word == word:
    print '%s\tnumber of users rated\t%s' % (current_count, current_word)
```

Hadoop Streaming Command

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.9.2.jar -D mapred.job.name=job_countByItemID -file program/mapper.py -mapper mapper.py -file program/reducer.py -reducer reducer.py -input /input/movies/u.data -output /output/movies/5
```

The job/application ID screenshot



Sample Output

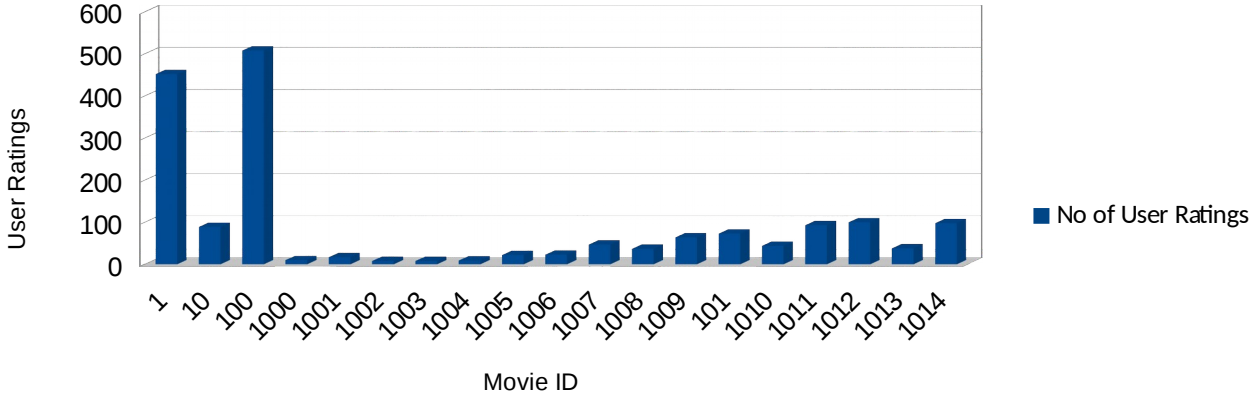
```
452 number of users rated    1
89  number of users rated    10
508 number of users rated    100
10  number of users rated    1000
17  number of users rated    1001
8   number of users rated    1002
8   number of users rated    1003
9   number of users rated    1004
22  number of users rated    1005
23  number of users rated    1006
47  number of users rated    1007
37  number of users rated    1008
```

```
64  number of users rated   1009
```

b) Navigate to the web UI for the history server (localhost:19888). Select the job with name "job_countByItemID" and then use the link on the left (as shown below) to navigate to the Counters. How many Map input records, Map output records, Reduce input records and Reduce output records do you see?

Map input records: 100,000
Map output records: 100,000
Reduce input records: 100,000
Reduce output records: 1,682

c) Save the result in a local directory and provide an Excel bar chart of only the first 20 movies (itemIDs) you see in the list, along with their frequencies (note that the output does NOT necessarily need to be sorted by popularity).



Exercise 2 – Counting popularity of movies (added combiner)

a) Consider about adding a combiner step on the previous streaming job.

- What is the purpose of adding a combiner?
  - Combiner is like a mini-reducer which reduces number of keys reducer need to process.
- Is it possible to construct a combiner function for the above problem? Justify your answer.
  - Above program is similar to wordcount program. We can construct combiner function for it. It takes input from mapper and gives output to the reducer. Here in our example we have similar keys with each mapper which can be reduced by combiner. Therefore, number of input records to the reducer can be reduced.
- What is the corresponding script for the combiner for this problem?
  - It can be similar to reducer script.

combiner.py

```
#!/usr/bin/python
```

```
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

b) Run the job again with having added a combiner and with the name of this job specified as “job_countByItemID_withCombiner”.

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.9.2.jar -D mapred.job.name=job_countByItemID_withCombiner -file program/mapper.py -mapper mapper.py -file program/combiner.py -combiner combiner.py -file program/reducer.py -reducer reducer.py -input /input/movies/u.data -output /output/movies/7
```

c) Inspect the Counters for this job and determine whether the combiner has been able to reduce the data transfer of key-value pairs between the mapper and the reducer phase. Provide necessary explanation.

Map input records: 100,000
Map output records: 100,000
Combiner input records: 100,000
Combiner output records: 3,159
Reduce input records: 3,159
Reduce output records: 1,682

It is clearly visible that number of records processed by reducer with combiner is lower than the one without combiner. Therefore, data transfer between nodes decreases which leads to better performance.

Exercise 3 – Histogram of movie counts

a) Based on the output of Exercise 1, write a python mapper and reducer script to "count the counts". In other words, you need to find how many movies have been rated once, how many twice, three times and so on? Output should be in the form of (frequency, number of movies with that frequency).

mapper.py

```
#!/usr/bin/python
import sys
for line in sys.stdin:
    line = line.strip()
    no_of_user, ignore, item_id = line.split('\t')
    print '%s\t%s' % (no_of_user, 1)
```

reducer.py

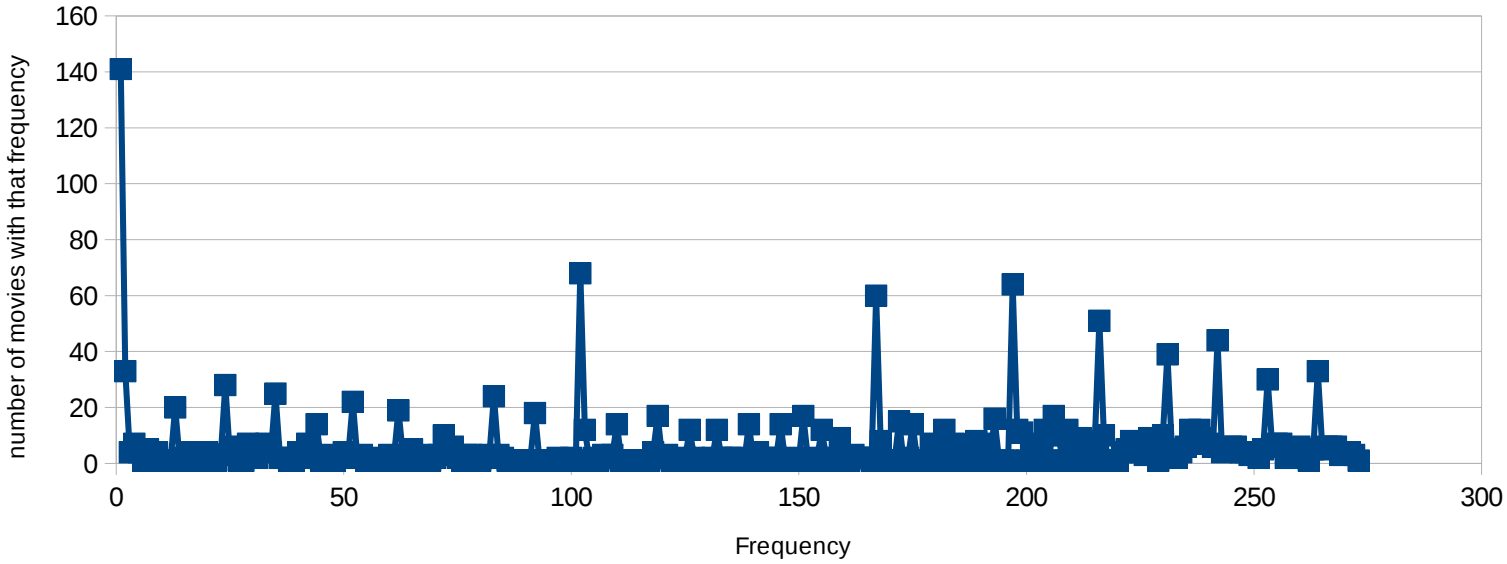
```
#!/usr/bin/python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Sample Output

```
1   141
10  33
100 4
101 7
102 4
```

```
103 1
104 5
105 1
106 4
107 1
108 1
109 1
```

b) Export the resulting pairs to a local directory and draw the respective scatterplot in Excel



- How many movies have been rated only once?
  - 140
- How many ratings did the most popular movie have?
  - 99

2. Exercises using Hive

Use Hive to answer the following questions. For all questions provide the hive scripts you have written.
Note for creating Hive managed tables: For the creation of the below tables you may use

- either the relevant datasets that are on your local drive C:/B1415/input/movies
- or the datasets that reside on your hdfs, in hdfs directory ./input/movies Remember that in this case, the relevant dataset will be moved from the hdfs directory user/yourName/input/movies to the hdfs hive directory in user/hive/warehouse.

Exercise 4 – Popularity of movies and average ratings

Write a hive query (or a set of hive queries) to:

a) create a managed table in Hive named uData to store all columns of the u.data dataset. The column names for uData should be userID, itemID, rating, unixtimestamp and they should all be specified as integers. Load the respective u.data into this and then display the first 20 records of the generated table.

- `CREATE TABLE uData (userid INT,movieid INT,rating INT,unixtime INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;`
- `LOAD DATA LOCAL INPATH '/home/hadoop/hadoop/dataset/movies/u.data' OVERWRITE INTO TABLE uData;`
- `select * from uData limit 0,20;`
- Output

```
OK
196 242 3   881250949
186 302 3   891717742
22  377 1   878887116
244 51  2   880606923
166 346 1   886397596
298 474 4   884182806
115 265 2   881171488
253 465 5   891628467
305 451 3   886324817
6   86  3   883603013
62  257 2   879372434
286 1014    5   879781125
200 222 5   876042340
210 40  3   891035994
224 29  3   888104457
303 785 3   879485318
122 387 5   879270459
194 274 2   879539794
291 1042    4   874834944
234 1184    2   892079237
Time taken: 0.11 seconds, Fetched: 20 row(s)
```

b) create a managed table in Hive named uItem to store the first two columns of the u.item dataset. The column names for uItem should be itemID, itemTitle, and they should be specified as integer and string respectively. Load the respective u.item into this and then display the first 20 records of the generated table.

- `CREATE TABLE uItem (itemID INT,itemTitle STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;`
- `LOAD DATA LOCAL INPATH '/home/hadoop/hadoop/dataset/movies/u.item' OVERWRITE INTO TABLE uItem;`
- `select * from uItem limit 0,20;`
- Output

```
OK
```

```
1   Toy Story (1995)
2   GoldenEye (1995)
3   Four Rooms (1995)
4   Get Shorty (1995)
5   Copycat (1995)
6   Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)
7   Twelve Monkeys (1995)
8   Babe (1995)
9   Dead Man Walking (1995)
10  Richard III (1995)
11  Seven (Se7en) (1995)
12  Usual Suspects, The (1995)
13  Mighty Aphrodite (1995)
14  Postino, Il (1994)
15  Mr. Holland's Opus (1995)
16  French Twist (Gazon maudit) (1995)
17  From Dusk Till Dawn (1996)
18  White Balloon, The (1995)
19  Antonia's Line (1995)
20  Angels and Insects (1995)
Time taken: 0.088 seconds, Fetched: 20 row(s)
```

c) join the uData table together with the uItem table in order to assign to all movie ids, their respective movie title. The resulting joined table should have the following columns: userID, itemID, rating, unixtimestamp, itemTitle. The joined table should be named uData_jn_Item. Display the first 20 records of this table.

- `CREATE TABLE uData_jn_Item as select d.userid, d.movieid, d.rating, d.unixtime, i.itemTitle from uData d inner join uItem i ON (d.movieid = i.itemID);`
- `select * from uData_jn_Item limit 0,20;`
- Output

```
OK
196 242 3   881250949  Kolya (1996)
186 302 3   891717742  L.A. Confidential (1997)
22  377 1   878887116  Heavyweights (1994)
244 51  2   880606923  Legends of the Fall (1994)
166 346 1   886397596  Jackie Brown (1997)
298 474 4   884182806  Dr. Strangelove or: How I Learned to Stop Worrying and Love
the Bomb (1963)
115 265 2   881171488  Hunt for Red October, The (1990)
253 465 5   891628467  Jungle Book, The (1994)
305 451 3   886324817  Grease (1978)
6   86  3   883603013  Remains of the Day, The (1993)
62  257 2   879372434  Men in Black (1997)
286 1014    5   879781125  Romy and Michele's High School Reunion (1997)
```

```
200 222 5   876042340  Star Trek: First Contact (1996)
210 40  3   891035994  To Wong Foo, Thanks for Everything! Julie Newmar (1995)
224 29  3   888104457  Batman Forever (1995)
303 785 3   879485318  Only You (1994)
122 387 5   879270459  Age of Innocence, The (1993)
194 274 2   879539794  Sabrina (1995)
291 1042    4   874834944  Just Cause (1995)
234 1184    2   892079237  Endless Summer 2, The (1994)
Time taken: 0.115 seconds, Fetched: 20 row(s)
```

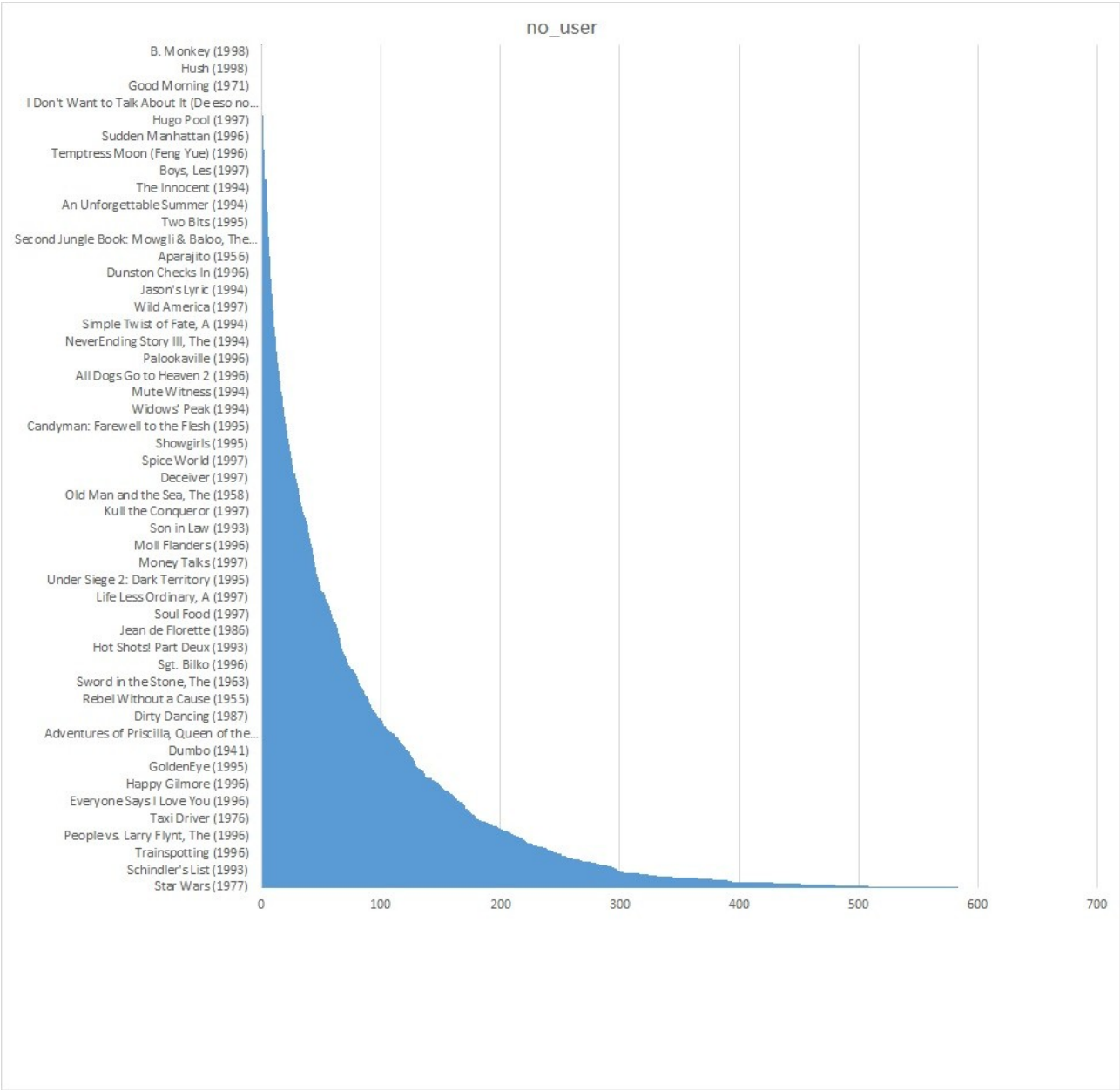
d) count the number of users that have rated each movie (which is an estimate of a movie's popularity), and sort them in popularity descending order.

```
create table ratings as select movieid, itemtitle, count(userid) as no_user from
uData_jn_Item group by movieid,itemtitle sort by no_user desc;
```

- How many movies have been rated at least 300 times

```
select count(movieid) from ratings where no_user > 299;
Output: 33
```

- Export the table into the local PC and draw a bar chart of the respective movie titles and their frequencies in Excel.



e) find the most-rated movie that has an average rating greater than or equal to 4.5?

```
select movieid, itemtitle, count(userid) as no_user, avg(rating) as avg_rating from uData_jn_Item group by movieid,itemtitle having avg_rating >= 4.5 sort by no_user desc;
```

```
Output:
OK
1449    Pather Panchali (1955)                          8   4.625
119     Maya Lin: A Strong Clear Vision (1994)          4   4.5
1293    Star Kid (1997)                                 3   5.0
1189    Prefontaine (1997)                              3   5.0
1594    Everest (1998)                                  2   4.5
1500    Santa with Muscles (1996)                       2   5.0
```

```
1467   Saint of Fort Washington, The (1993)                2   5.0
1642   Some Mother's Son (1996)                             2   4.5
1398   Anna (1996)                                          2   4.5
1599   Someone Else's America (1995)                        1   5.0
1536   Aiqing wansui (1994)                                 1   5.0
1201   Marlene Dietrich: Shadow and Light (1996)            1   5.0
1122   They Made Me a Criminal (1939)                       1   5.0
814    Great Day in Harlem, A (1994)                        1   5.0
1653   Entertaining Angels: The Dorothy Day Story (1996)    1   5.0
Time taken: 30.086 seconds, Fetched: 15 row(s)
```

Exercise 5 – Breakdown of movies by gender

Write a hive query (or a set of hive queries) to:

a) create a managed table in Hive named uUser to store all columns of the u.user dataset. The column names for uUser should be userID, age, gender, job, zipCode, where userID, age, zipCode are all integers, while gender and job are strings. Load the respective u.user into this and then display the first 20 records of this table.

- `CREATE TABLE uUser (userID INT,age INT,gender STRING,job STRING,zipCode INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;`
- `LOAD DATA LOCAL INPATH '/home/hadoop/hadoop/dataset/movies/u.user' OVERWRITE INTO TABLE uUser;`
- `select * from uUser limit 0,20;`
- Output:

```
OK
1   24  M   technician    85711
2   53  F   other         94043
3   23  M   writer        32067
4   24  M   technician    43537
5   33  F   other         15213
6   42  M   executive     98101
7   57  M   administrator 91344
8   36  M   administrator 5201
9   29  M   student       1002
10  53  M   lawyer        90703
11  39  F   other         30329
12  28  F   other         6405
13  47  M   educator      29206
14  45  M   scientist     55106
15  49  F   educator      97301
16  21  M   entertainment 10309
17  30  M   programmer    6355
18  35  F   other         37212
19  40  M   librarian     2138
20  42  F   homemaker     95660
Time taken: 0.11 seconds, Fetched: 20 row(s)
```

b) to join the uData_jn_Item table together with the uUser table in order to assign to all users, their respective gender. The new table should have the following columns: userID, itemID, rating, unixtimestamp, itemTitle, gender. Display the first 20 records of this table.

- `CREATE TABLE uData_with_Gender as select u.userid, d.movieid, d.rating, d.unixtime, d.itemTitle, u.gender from uData_jn_Item d inner join uUser u ON (d.userid = u.userid);`
- `select * from uData_with_Gender limit 0,20;`
- Output

```
OK
196 242 3   881250949  Kolya (1996)  M
186 302 3   891717742  L.A. Confidential (1997) F
22  377 1   878887116  Heavyweights (1994)  M
244 51  2   880606923  Legends of the Fall (1994)  M
166 346 1   886397596  Jackie Brown (1997)  M
298 474 4   884182806  Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)  M
115 265 2   881171488  Hunt for Red October, The (1990)  M
253 465 5   891628467  Jungle Book, The (1994)  F
305 451 3   886324817  Grease (1978) M
6   86  3   883603013  Remains of the Day, The (1993)  M
62  257 2   879372434  Men in Black (1997)  F
286 1014    5   879781125  Romy and Michele's High School Reunion (1997) M
200 222 5   876042340  Star Trek: First Contact (1996) M
210 40  3   891035994  To Wong Foo, Thanks for Everything! Julie Newmar (1995) M
224 29  3   888104457  Batman Forever (1995)  F
303 785 3   879485318  Only You (1994)  M
122 387 5   879270459  Age of Innocence, The (1993)  F
194 274 2   879539794  Sabrina (1995)  M
291 1042    4   874834944  Just Cause (1995) M
234 1184    2   892079237  Endless Summer 2, The (1994)  M
Time taken: 0.119 seconds, Fetched: 20 row(s)
```

c) display the breakdown of number of records by men (M) and women (F), and also the average rating provided for each gender.

```
select gender, count(userid) as no_user, avg(rating) as avg_rating from uData_with_Gender group by gender;
```

- Are most ratings provided by men or women?
  - men
- Does their average rating provided by men and women differ substantially?
  - No
- Output

```
OK
F   25740  3.5315073815073816
M   74260  3.5292889846485322
Time taken: 13.119 seconds, Fetched: 2 row(s)
```

d) find the most-rated movie for men that has an average rating of 5, and which for women?

- `select movieid, itemtitle, count(userid) as no_user, avg(rating) as avg_rating from uData_with_Gender where gender = 'M' group by movieid,itemtitle having avg_rating = 5 sort by no_user desc;`
- Output:

```
OK
1293   Star Kid (1997)   3   5.0
1175   Hugo Pool (1997)  2   5.0
1612   Leading Man, The (1996)  2   5.0
1500   Santa with Muscles (1996)   2   5.0
1467   Saint of Fort Washington, The (1993)   2   5.0
1189   Prefontaine (1997)   2   5.0
1656   Little City (1998)   1   5.0
1653   Entertaining Angels: The Dorothy Day Story (1996)    1   5.0
1605   Love Serenade (1996) 1   5.0
1536   Aiqing wansui (1994) 1   5.0
1306   Delta of Venus (1994)    1   5.0
1201   Marlene Dietrich: Shadow and Light (1996)     1   5.0
1191   Letter From Death Row, A (1998) 1   5.0
1144   Quiet Room, The (1996)   1   5.0
1122   They Made Me a Criminal (1939)  1   5.0
814 Great Day in Harlem, A (1994)   1   5.0
Time taken: 29.079 seconds, Fetched: 16 row(s)
```

- `select movieid, itemtitle, count(userid) as no_user, avg(rating) as avg_rating from uData_with_Gender where gender = 'F' group by movieid,itemtitle having avg_rating = 5 sort by no_user desc;`
- Output:

```
OK
1368   Mina Tannenbaum (1994)   2   5.0
1599   Someone Else's America (1995)   1   5.0
1594   Everest (1998)    1   5.0
1472   Visitors, The (Visiteurs, Les) (1993)  1   5.0
1451   Foreign Correspondent (1940)    1   5.0
1301   Stripes (1981)    1   5.0
1189   Prefontaine (1997)   1   5.0
884 Year of the Horse (1997) 1   5.0
883 Telling Lies in America (1997)  1   5.0
119 Maya Lin: A Strong Clear Vision (1994) 1   5.0
74  Faster Pussycat! Kill! Kill! (1965)    1   5.0
Time taken: 30.235 seconds, Fetched: 11 row(s)
```

3. Exercises in Pig

Use Pig to do the following tasks. For all exercises provide the pig statements you have written.

Exercise 6 – Find the top 20 movies by average rating score

Write a pig statement sentence to:

a) load u.data and specify a schema of four columns named userID, itemID, rating, unix_timestamp all specified as integers

```
u_data = LOAD 'hdfs://aai07server:9000/input/movies/u.data' USING PigStorage('\t') as (userID:int, itemID:int, rating:int, unix_timestamp:int);
```

b) to group the relation in (a) by the itemID field

```
u_data_group = GROUP u_data BY itemID;
```

c) to calculate the average rating score for each movie based on the relation in (b)

```
avg_rating = FOREACH u_data_group GENERATE FLATTEN(u_data.itemID), AVG(u_data.rating) as avg_u_data;
```

d) to sort the previous relation (c) in descending order

```
avg_rating_sort = ORDER avg_rating BY u_data.itemID DESC;
```

e) to keep only the top 20 movie ids from the relation in (d)

```
limit_avg_rating = LIMIT avg_rating_sort 20;
```

f) execute data flow and instruct pig to store the relation in (e) on HDFS. Provide a list of the top 20 movies ids and their average rating score.

```
STORE limit_avg_rating INTO 'hdfs://aai07server:9000/output/movies/pig_Output' USING PigStorage (',');
```

Output:

```
1682    3
1681    3
1680    2
1679    3
1678    1
1677    3
1676    2
1675    3
1674    4
1673    3
1672    2
1672    2
1671    1
1670    3
1669    2
1668    3
1667    3
1666    2
1665    2
1664    3.25
```

Exercise 7 – Find the worst movies based on average rating score

Write a pig statement sentence to:

a) to keep only movies with average rating score not greater than 1.5.

```
filter_data = FILTER avg_rating BY avg_u_data <= 1.5;
```

b) execute data flow and store the result on HDFS. How many movies are there?

```
filter_group = GROUP filter_data BY itemID;
count_item = FOREACH filter_group GENERATE COUNT(filter_data.avg_u_data) as
count_data;
count_group = GROUP count_item ALL;
total_item = FOREACH count_group GENERATE SUM(count_item.count_data);
STORE total_item INTO 'hdfs://aai07server:9000/output/movies/pig_total_item';

Output: 170
```