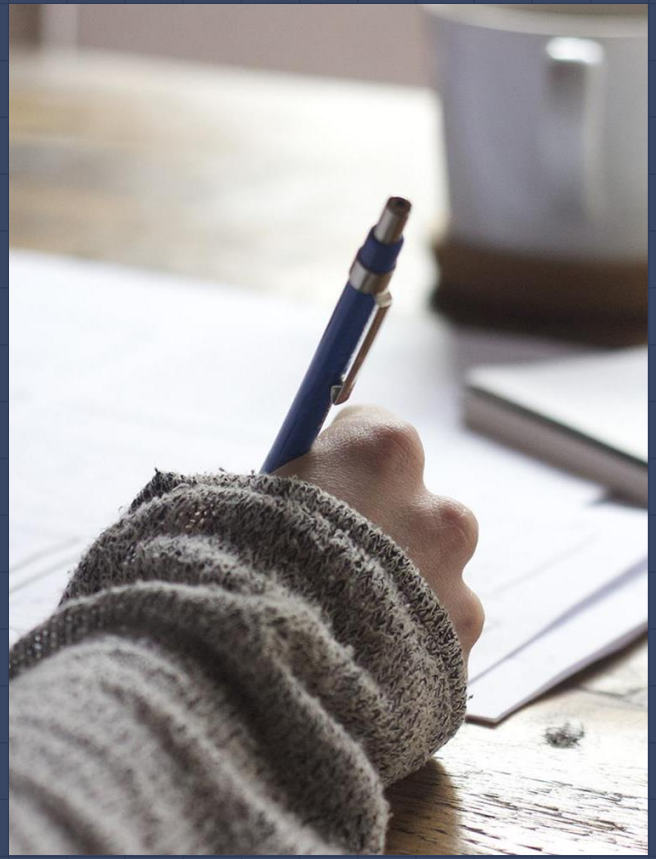
HELLO!

I am MJ Sadeghi

Ms data sicence and analytics

mjs198867@gmail.com



Dataset

Let’s start with the first set of slides

1

Datasets

- Data set can be found in MovieLens (https://grouplens.org/datasets/movielens)
- It includes 100,000 ratings (1-5) from 943 users on 1682 movies . Each user has rated at least 20 movies.
- The data was collected through the MovieLens web site

  (movielens.umn.edu) during the seven-month

  period from September 19th, 1997 through

  April 22nd, 1998.
- Simple demographic info for the users (age, gender, occupation, zip)

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

u.data content

```
movie id | movie title | release date | video
IMDb URL | unknown | Action | Adventure | Animation
Children's | Comedy | Crime | Documentary | Drama |
Film-Noir | Horror | Musical | Mystery | Romance |
Thriller | War | Western |
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?
Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?
GoldenEye%20(1995)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-
exact?Four%20Rooms%20(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-
exact?Get%20Shorty%20(1995)|0|1|0|0|0|1|0|0|1|0|0|0|0|0|0|0|0
```

u.item content

Why there is a need for map-reduce ?!!

When this data set has

943 users

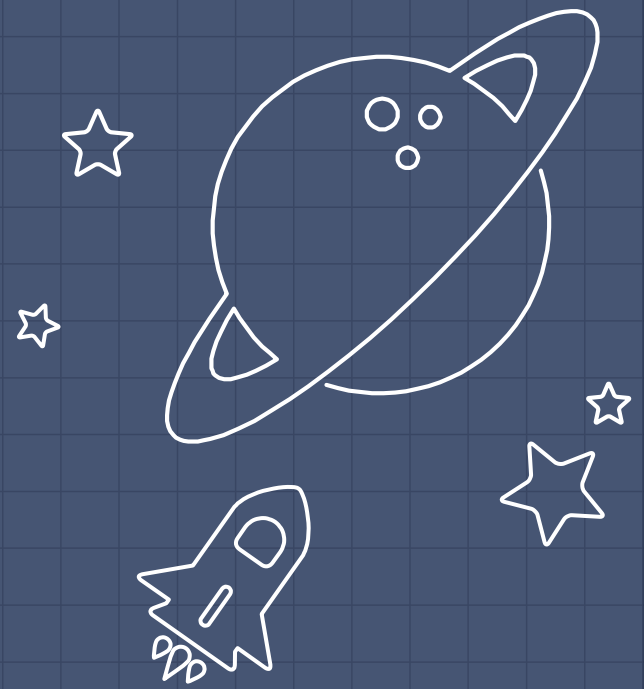
There are plenty of users all around the world

1682 Movies

How many movies are available ?!!

100,000 Records

How big our data set can be ?!!



What if 100,000,000 users all around the world rank all movies ?!!

Implementation and results

Let’s start with the second set of slides

2

Implementation

Using Hortonworks Sandbox HDP 2.6.5

It provides the unix shell in your browser (localhost:4200).

It also provides an Ambari Inteface to deal with different Hadoop Eco-system technologies easily

Using Python

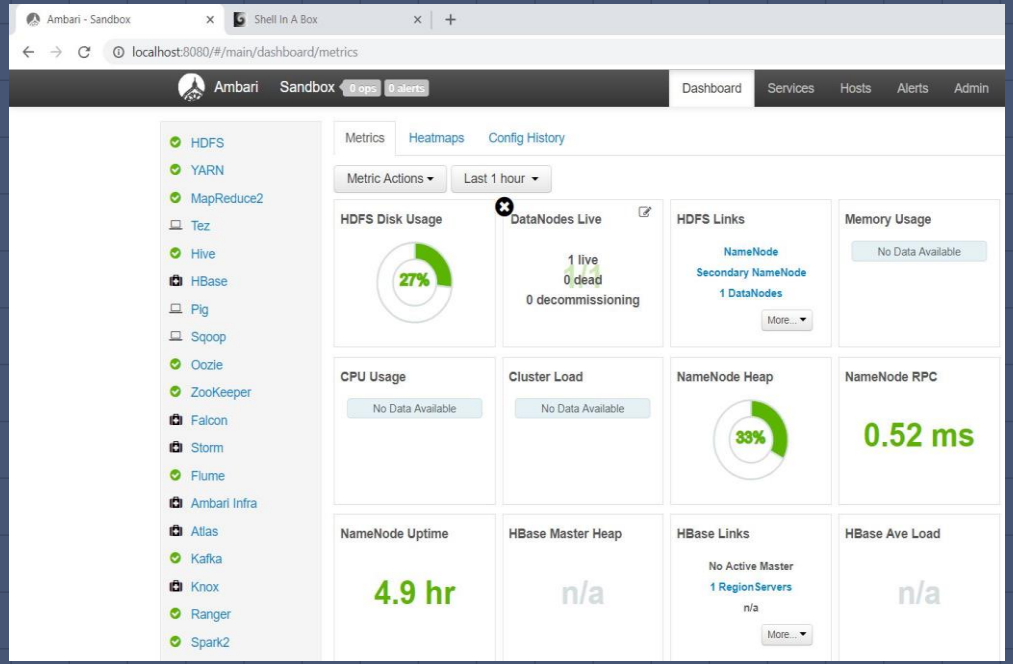
Using pip to install MRJob and required packages.

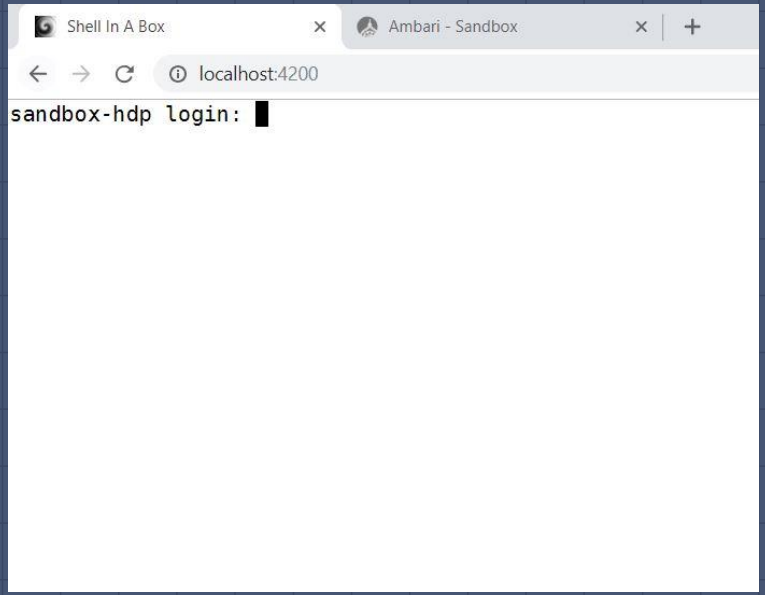
Using MRJob to code in python for map reduce.

Hadoop Eco-system and R

After using map reduce to generate our new data set, we can analyze this data set by Hive(using SQL), R and so on.

Hortonworks Sandbox and the Ambari Enviroment





The Ambari interface that accessable from localhost:8080 through the browser. The available services can be seen on the left.

The shell in a browserLocalhost:4200

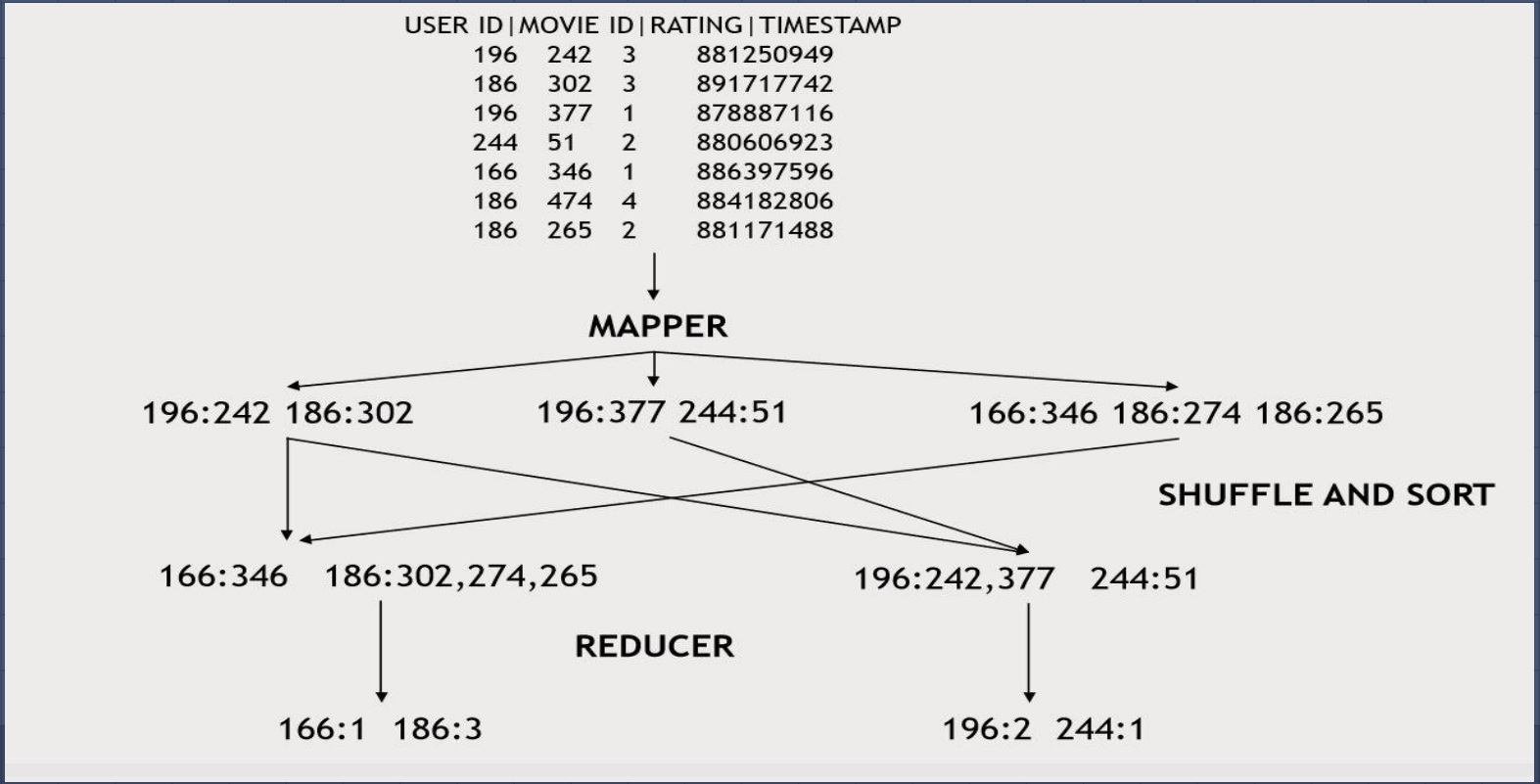
python

- Most of programs for this madule were written by Python programming language.
- Python can handle the map-reduce job by using a library called MRJob easily .

Why python?

- ✓ It is open source, And there are bunch of libraries and opened sources program on the Internet(Github) for most tasks that you are dealing with.
- ✓ It is easier to write code with in compare to Java
- ✓ Python is one the most demand programming language in companies or industries related to data science.
- ✓ It is kind of the main programming language for AI and Mechine Learning.

Program 1: Map-Reduce Diagram



Map-Reduce Program 1

How many people ranked 5 stars
How many people ranked 4 stars
…
How many people ranked 1 stars

Using python and map-reduce
Using mr-job library

Input data:

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

Goals

Braking down the ranking based on stars

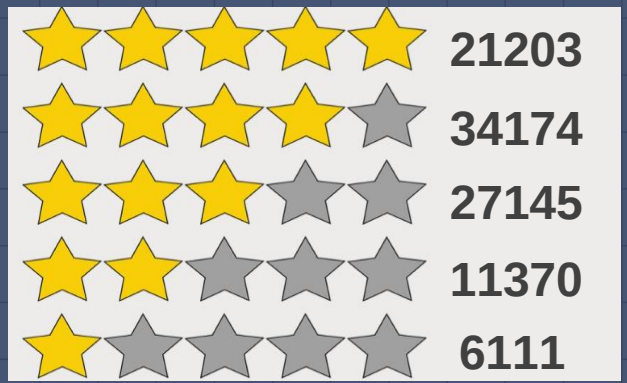
Program 1. numbers of people who ranked all movies from 1 to 5 stars

File name : Ratings.py

```python
from mrjob.job import MRJob
from mrjob.step import MRStep

class Ratings(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_ratings,
                   reducer=self.reducer_ratings)
        ]

    def mapper_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer_ratings(self, ranking, counts):
        yield ranking, sum(counts)


if __name__ == '__main__':
    Ratings.run()
```

result

```
                Spilled Records=200006
                Total committed heap usage (bytes)=240123904
                Virtual memory (bytes) snapshot=5845512192
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
Streaming final output from hdfs:///user/maria_dev/tmp/mrjob/Ratings.
"1"     6111
"2"     11370
"3"     27145
"4"     34174
"5"     21203
Removing HDFS temp directory hdfs:///user/maria_dev/tmp/mrjob/Ratings
Removing temp directory /tmp/Ratings.maria_dev.20181201.164637.205552
```



By executing this code in Hadoop in shell :

```
python Ratings.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar u.data
```

In program 2, I explain how we can write a program for map-reduce in python. So every program can easily be understood

Map-Reduce Program 2

Input data:

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

Goals

Specifying the number of people who ranked every movies.

Program 2. How many votes each movie received totally(all users) ordered by number of votes

File name : RatingsSum.py

```
from mrjob.job import MRJob
from mrjob.step import MRStep


class RatingsSum(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_ratings,
                   reducer=self.reducer_first),
            MRStep(reducer=self.reducer_last)
        ]

    def mapper_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield movieID, 1


    def reducer_first(self, key, values):
        yield str(sum(values)).zfill(5), key


    def reducer_last(self, count, movies):
        val=""
        for value in movies:
            val=str(value)+","+val
        yield count,val

if __name__ == '__main__':
    RatingsSum.run()
```

At first, required libraries have been imported(MRJob and MRStep). In python, def is a word to write a function.

In steps, mapper and reducer were defined. In mapper_ratings(our mapper), line(our data set line by line) would be separated by delimiter “\t”(tab) and put in the userID , MovieID , rating and timestamp variables respectively. Yield specify the output of our mapper or reducer. In this case, we want to see how many votes every movie received, so movieID and 1 is our output.

In reducer_first , “key = movieID” and “values” is the list of 1 from the yield of first reducer, the sum(values) and key(movieID) would sent to the second mapper to add up all the movieID’s with the same number of votes.

By executing this code in Hadoop (shell) :

```
python RatingsSum.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar u.data > RatingSum.txt
```

Result 2 >>> How many votes each movie received totally(by all users) ordered by number of votes

"00001"
"857,852,830,814,711,677,599,1682,1681,1680,1679,1678,1677,1676,1675,1674,1673,1671,1670,1669,1668,1667,1666,1665,1663,1661,1660,1659,1657,1655,1654,1653,1651,1650,1649,1648,1647,1645,1641,1640,1638,1637,1636,1635,1634,1633,1632,1630,1627,1626,1625,1624,1621,1619,1618,1616,1614,1613,1606,1604,1603,1601,1599,1596,1595,1593,1587,1586,1584,1583,1582,1581,1580,1579,1577,1576,1575,1574,1572,1571,1570,1569,1568,1567,1566,1565,1564,1563,1562,1561,1559,1557,1548,1546,1543,1536,1533,1526,1525,1520,1515,1510,1507,1505,1498,1494,1493,1492,1486,1482,1476,1461,1460,1458,1457,1453,1452,1447,1414,1373,1366,1364,1363,1352,1349,1348,1343,1341,1340,1339,1329,1325,1320,1310,1309,1236,1235,1201,1156,1130,1122,“

"00002"
"957,913,907,897,784,600,1672,1662,1656,1646,1644,1642,1631,1629,1617,1611,1594,1590,1588,1585,1578,1573,1556,1554,1551,1550,1549,1547,1542,1541,1532,1523,1519,1502,1500,1497,1491,1481,1477,1472,1467,1455,1450,1436,1433,1417,1398,1390,1374,1371,1365,1362,1360,1359,1358,1354,1350,1345,1342,1334,1332,1321,1308,1307,1306,1304,1290,1080,“

"00003"
"861,858,817,788,1658,1652,1639,1623,1622,1610,1609,1607,1602,1552,1544,1538,1528,1516,1513,1508,1506,1504,1490,1484,1465,1464,1463,1448,1432,1430,1424,1420,1418,1408,1391,1389,1387,1384,1372,1370,1361,1357,1356,1351,1327,1323,1318,1293,1292,1256,1196,1191,1189,1186,1155,1146,1144,1123,1096,1027,"
"00004“

"00429" "121,"
"00431" "300,"
"00452" "1,"
"00478" "288,"
"00481" "286,"
"00485" "294,"
"00507" "181,"
"00508" "100,"
"00509" "258,"
"00583" "50,"

First column(in light blue) is the vote number.

Second column numbers(in black) separated by “,” are the correlative movieIDs.

Map-Reduce Program 3

Goals

Movie ratings

Input data:

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

Program 3. how many one ,two ,three ,four and five star votes every movie received

File name : MovieStars.py

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class MovieStars(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_ratings,
                   reducer=self.reducer_ratings),
            MRStep(reducer=self.reducer_last)
        ]

    def mapper_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield movieID,float(rating)

    def reducer_ratings(self, movieID, rating):
        values=""
        a=","
        i=j=e=f=d=0
        for value in rating:
            if value==1:
                i=i+1
            elif value==2:
                j=j+1
            elif value==3:
                f=f+1
            elif value==4:
                e=e+1
            elif value==5:
                d=d+1
            values="1star="+str(i)+a+"2stars="+str(j)+a+"3stars="+str(f)+a+"4stars="+str(e)+a+"5stars="+str(d)

        yield str(movieID).zfill(5), values

    def reducer_last(self, movieID, stars):
        val=""
        for value in stars:
            val=str(value)
        yield movieID,val


if __name__ == '__main__':
    MovieStars.run()
```

Most part of how a map-reduce program can be written was explained in program 2.

Here there is a if - elif that works like if - else in Java. Str() is a casting function (to string) and zfill(n) puts 0, n times before our int, in our case zfill(5) would put 5 zero before the ingeger to make it sortable for 2nd reducer, all the other logic of programing roughly the same in every object oriented programming

By executing this code in Hadoop (shell) :

```
python MovieStars.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar u.data
```

Result of program 3 >>> how many one ,two ,three ,four and five star votes every movie received

"00001"	"1star=8,2stars=27,3stars=96,4stars=202,5stars=119"
"00002"	"1star=8,2stars=17,3stars=55,4stars=42,5stars=9"
"00003"	"1star=11,2stars=20,3stars=25,4stars=23,5stars=11"
"00004"	"1star=6,2stars=24,3stars=57,4stars=93,5stars=29"
"00005"	"1star=4,2stars=11,3stars=32,4stars=33,5stars=6"
"00006"	"1star=2,2stars=4,3stars=5,4stars=7,5stars=8"
"00007"	"1star=9,2stars=31,3stars=89,4stars=164,5stars=99"
"00008"	"1star=4,2stars=13,3stars=47,4stars=71,5stars=84"
"00009"	"1star=11,2stars=18,3stars=59,4stars=114,5stars=97"
"00010"	"1star=2,2stars=7,3stars=21,4stars=33,5stars=26"
"00011"	"1star=8,2stars=23,3stars=38,4stars=95,5stars=72"
...	...
"00044"	"1star=6,2stars=6,3stars=29,4stars=31,5stars=7"
"00045"	"1star=1,2stars=1,3stars=20,4stars=29,5stars=29"
"00046"	"1star=1,2stars=2,3stars=8,4stars=13,5stars=3"
"00047"	"1star=5,2stars=17,3stars=32,4stars=51,5stars=28"
"00048"	"1star=3,2stars=7,3stars=11,4stars=51,5stars=45"
"00049"	"1star=2,2stars=10,3stars=36,4stars=26,5stars=7"
"00050"	"1star=9,2stars=16,3stars=57,4stars=176,5stars=325"
"00051"	"1star=6,2stars=11,3stars=19,4stars=30,5stars=15"
"00052"	"1star=1,2stars=5,3stars=29,4stars=35,5stars=21"
"00053"	"1star=26,2stars=21,3stars=30,4stars=35,5stars=16"
"00054"	"1star=1,2stars=18,3stars=45,4stars=35,5stars=5“
...	...
"01677"	"1star=0,2stars=0,3stars=1,4stars=0,5stars=0"
"01678"	"1star=1,2stars=0,3stars=0,4stars=0,5stars=0"
"01679"	"1star=0,2stars=0,3stars=1,4stars=0,5stars=0"
"01680"	"1star=0,2stars=1,3stars=0,4stars=0,5stars=0"
"01681"	"1star=0,2stars=0,3stars=1,4stars=0,5stars=0"
"01682"	"1star=0,2stars=0,3stars=1,4stars=0,5stars=0"

First column(in red) is movieIDs.

Map-Reduce Program 4

Modifying the map-reduce program 3 and calculating average ratings

Goals

Average ratings
Having movie ratings in usable format

Input data:

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

Program 4. how many one ,two ,three ,four and five star votes every movie received(modified)+ avg rating

File name : MovieStars_avg.py

```python
from mrjob.job import MRJob
from mrjob.step import MRStep

class MovieStars(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_ratings,
                   reducer=self.reducer_ratings),
            MRStep(reducer=self.reducer_last)
        ]

    def mapper_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield movieID,float(rating)

    def reducer_ratings(self, movieID, rating):
        values=""
        a=one=two=th=four=five=0
        avgRating=float(a)
        for value in rating:
            if value==1:
                one=one+1
            elif value==2:
                two=two+1
            elif value==3:
                th=th+1
            elif value==4:
                four=four+1
            elif value==5:
                five=five+1
        values = str(one)+"    "+str(two)+"    "+str(th)+"    "+str(four)+"    "+str(five)
        avgRating=float(one*1+ two*2 +th*3+ four*4+ five*5)/(one+two+th+four+five)
        values=values+'    '+str(avgRating)
        yield int(movieID),values

    def reducer_last(self, movieID, stars):
        val=""
        for value in stars:
            val=value
        yield movieID,val


if __name__ == '__main__':
    MovieStars.run()
```

After modifying the program 3, average ratings added up to program 4.

First Result

```
1       "8    27    96    202    119    3.87831858407"
10      "2    7     21    33     26     3.83146067416"
100     "14    18     70     179     227     4.15551181102"
1000    "0    2     6     2     0     3.0"
1001    "10    2     1     3     1     2.0"
1002    "5    0     2     1     0     1.875"
```

From left to right columns are movieID , number of one stars, two stars, three stars, four stars, five stars votes and average movie ranking respectively.

Using little peace of code below to put the data in a way to be ready for use :

```python
stars=open("C:/Users/M/Desktop/HPCI-PRoj/test1_order.txt","r")
movieStars=open("C:/Users/M/Desktop/HPCI-PRoj/MovieStars_avg.txt","a")
s=","
for x in stars:
    stars_clean=x.replace('"','').split("\t")
    a=stars_clean[1].split("    ")
    data=stars_clean[0]+s+a[0]+s+a[1]+s+a[2]+s+a[3]+s+a[4]+s+a[5]+'\n'
    movieStars.write(data)
```

Final Result

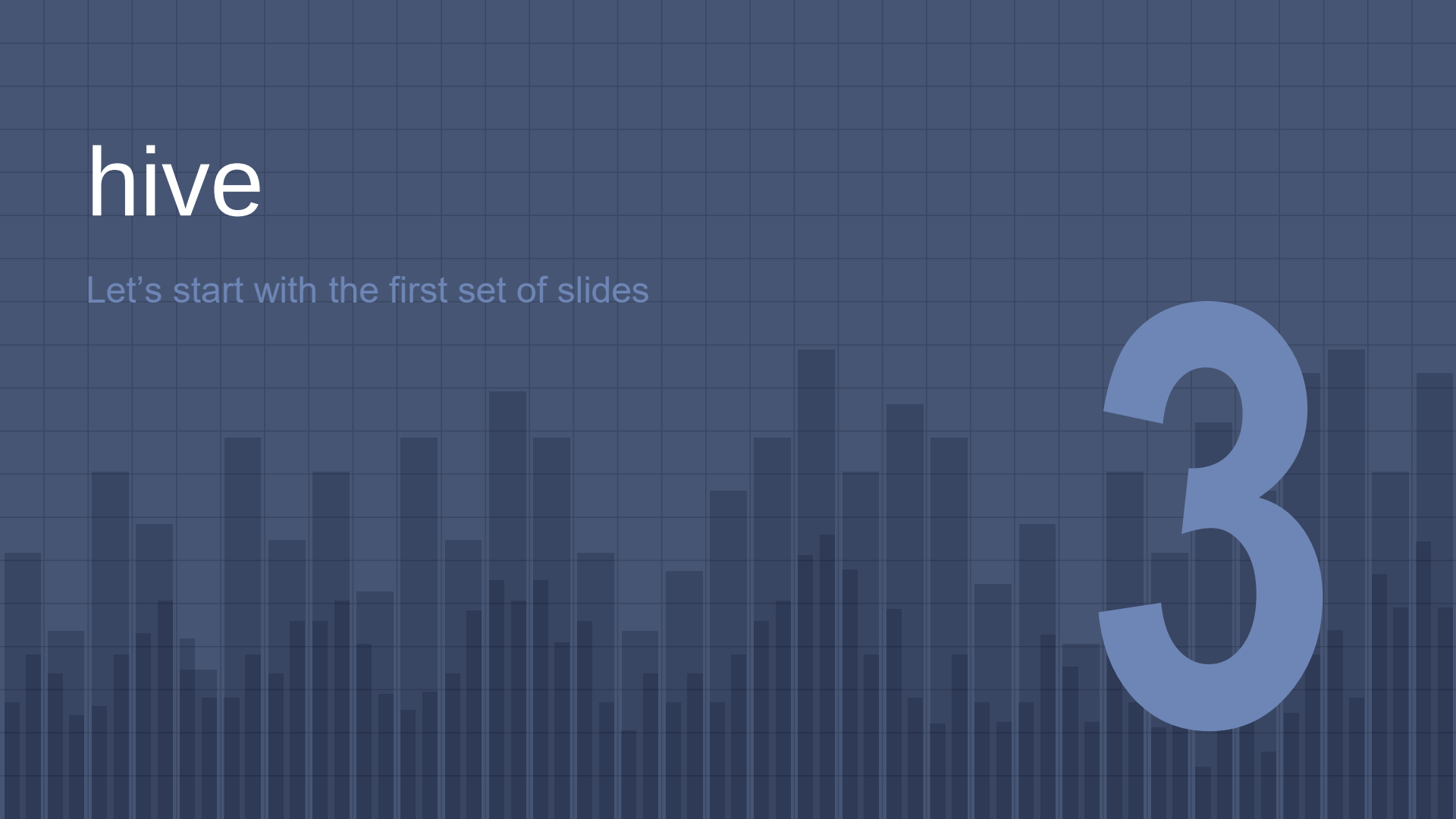
```
1,8,27,96,202,119,3.87831858407

10,2,7,21,33,26,3.83146067416

100,14,18,70,179,227,4.15551181102

1000,0,2,6,2,0,3.0
```

By executing this code in Hadoop (shell) :

```
python MovieStars_avg.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar u.data
```

hive

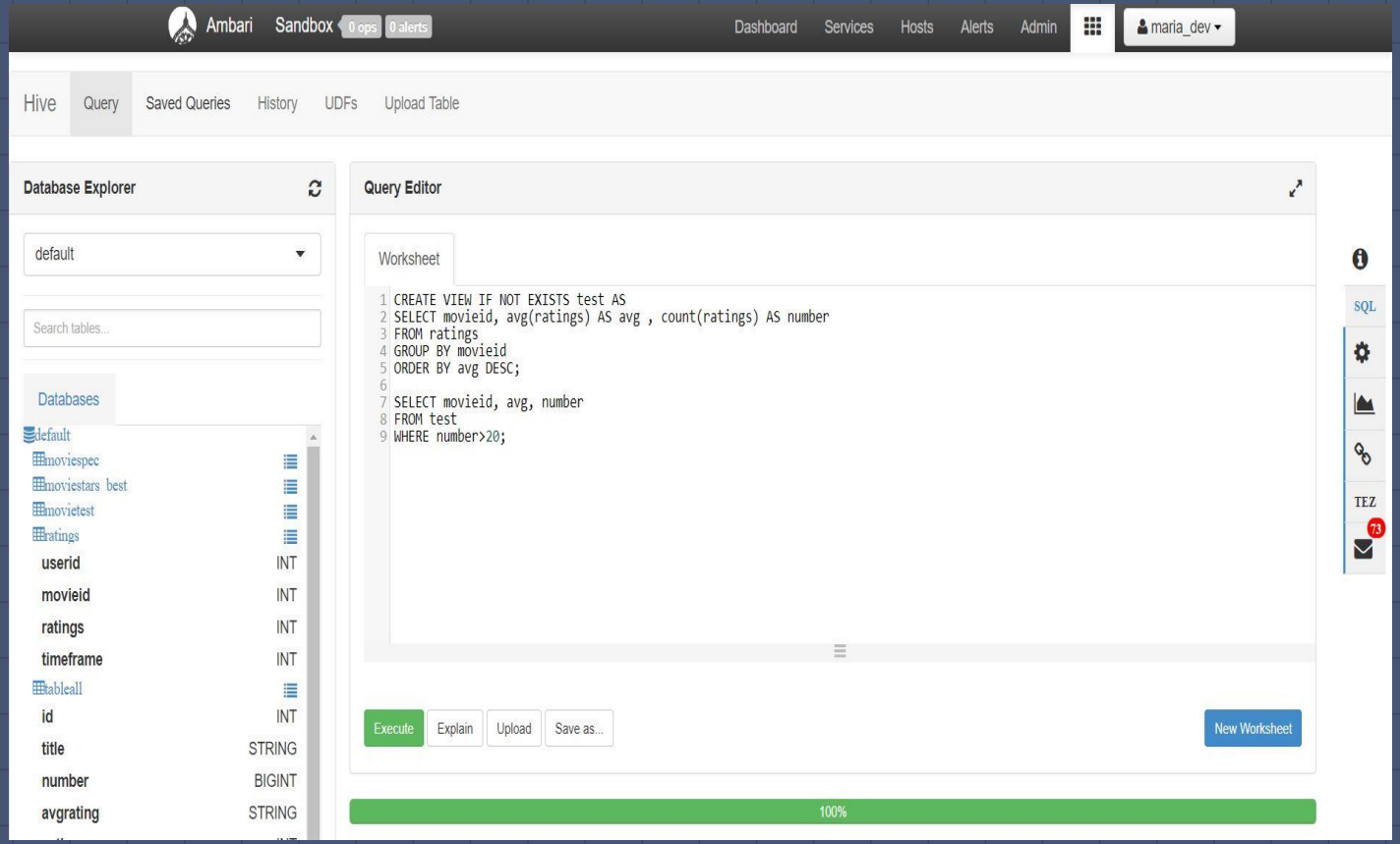
Let’s start with the first set of slides



Hive

- Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis.
- Hive gives a SQL-like interfaceto query data stored in various databases and file systems that integrate with Hadoop.
- Traditional SQL queries must be implemented in the MapReduceJava API to executeSQL applications and queries over distributed data. Hive provides the necessary SQL abstraction to integrate SQL-like queries (HiveQL) into the underlying Javawithout the need to implement queries in the low-level Javac

Hive Enviroment in Ambari



This the hive environment. As it can be seen from image, there is Database section at left where tables' structures can be seen.

There is Query Editor that SQL like programming can be done at.

A green button at the bottom of the page to execute our code (query).

Datasets using in Hive

1. Moviespec table :u.Item content , the table is opened in R to make it easy to see the content of u.item file(dataset).

	movieID	movieTitle	T	IMDB	Unknown	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	FilmNoir	Horror
1	Toy Story (1995)	01-Jan-1995	NA	http://...	0	0	0	1	1	1	0	0	0	0	0	
2	GoldenEye (19...	01-Jan-1995	NA	http://...	0	1	1	0	0	0	0	0	0	0	0	
3	Four Rooms (1...	01-Jan-1995	NA	http://...	0	0	0	0	0	0	0	0	0	0	0	
4	Get Shorty (19...	01-Jan-1995	NA	http://...	0	1	0	0	0	1	0	0	1	0	0	
5	Copycat (1995)	01-Jan-1995	NA	http://...	0	0	0	0	0	0	1	0	1	0	0	
6	Shanghai Triad...	01-Jan-1995	NA	http://...	0	0	0	0	0	0	0	0	1	0	0	
7	Twelve Monke...	01-Jan-1995	NA	http://...	0	0	0	0	0	0	0	0	1	0	0	

R code

```
> setwd("C:\\Users\\M\\Desktop\\HPCI-PRoj")        # set the work directory(in where u.item is)
> t=read.table("u.item",header = T,sep = "|",quote = "")    # using read.table to read table, using “|”delimiter
```

2. Moviestars_best table : Result of program 4

left to right columns are movieID , number ofone stars, two stars, three stars, four stars, five stars votes and average movie ranking respectively.

	movieID	oneStars	twoStars	threeStars	fourStars	FiveStars	avg
1	1	8	27	96	202	119	3.878319
2	10	2	7	21	33	26	3.831461
3	100	14	18	70	179	227	4.155512
4	1000	0	2	6	2	0	3.000000
5	1001	10	2	1	3	1	2.000000

R code > read.table(“MovieStars_best.item", header = T, sep = “,")
using read.table to read table, using “,” delimiter

3. Ratings table : Content of u.data

Uploaded in hive table with the ratings name. 100k rows

	userID	movieID	ratings	timeStamp
1	196	242	3	881250949
2	186	302	3	891717742
3	22	377	1	878887116
4	244	51	2	880606923
5	166	346	1	886397596

R code > read.table("u.data", header = T, sep = “\t")
using read.table to read table, using “\t” delimiter which is Tab

Hive Program 1

Joining all the table together with desired field

Tables:

1. Moviespec table
2. Moviestars_best table
3. Ratings table

Result**:**

1. alltable

Goals

It is going to be more straightforward and much easier to make a query on.

Hive Program 1. Joining all the three tables

Worksheet *

1st part

First pat explanation ➔

```
1  CREATE VIEW IF NOT EXISTS ratings_filter AS
2  SELECT movieid, count(ratings) AS number
3  From ratings
4  GROUP BY movieid
5  ORDER By number DESC;
6
7  CREATE VIEW IF NOT EXISTS alltable AS
8  SELECT r.movieid AS id , m.movietitle AS title , mb.avgrating AS avg , number , m.action , m.comedy , m.animation , m.drama ,
9  m.horror, mb.onestars , mb.twostars , mb.threestars , mb.fourstars , mb.fivestars
10 FROM (( ratings_filter r JOIN moviespec m ON r.movieid=m.movieid)JOIN moviestars_best mb ON r.movieid=mb.movieid);
```

2nd part

2nd part :

Another view named all table is created. Selecting movieID (r.movieid=raitings.movieid) From the raitings_filter view which was created in 1st part. By ”m.movietitle”, movietitle is selected from movie table, and using “AS title” to specifty a new name when it will save to alltable . The process of selecting other values is the same (mb point to the moviestars_best table).

In From section, First, ratings_filter(view),alias as r, and moviespec table(alias as m) join toghether based on movieids that are common in two table, and after that they will join with moviestars_best(mb).

The below result shows how alltable is like. Now, we are able to query on all table to get what we want.

Code explanation is broke down to two parts.

1st part :

Using SQL “CREATE VIEW” to make a new view(table) named ratings_filter which we can query on it.

Second line woks like map reduce, using “SELECT” we choose the movieID and using bult-in function count() to receive length of ratings.

” GROUP BY” to say we want to map all the ratings based on movieID. “From” specify the table on which is queried.

Finally, we put all the result in order using ”ORDER” command in Descending way by “DESC”.

A sample of result:

	alltable.id	alltable.title	alltable.avg	alltable.number	alltable.action	alltable.comedy	alltable.animation	alltable.drama	alltable.horror	alltable.onestars	alltable.twostars	alltable.threestar	alltable.fourstar	alltable.fivestars
2	50	Star Wars (1	4.3584906	583	1	0	0	0	0	9	16	57	176	325
3	258	Contact (19	3.8035363	509	0	0	0	1	0	12	43	110	212	132
4	100	Fargo (1996	4.1555118	508	0	0	0	1	0	14	18	70	179	227
5	181	Return of th	4.0078895	507	1	0	0	0	0	8	23	97	208	171
6	294	Liar Liar (19	3.156701	485	0	1	0	0	0	47	73	168	151	46
7	286	English Patie	3.6569647	481	0	0	0	1	0	30	53	100	167	131

Hive Program 2

Comparing average

Tables:

1. New view named filter_ratings
2. alltable

Result**:**

Selected query

Goals

Comparing average from map-reduce program 4 to average calculated by Hive to see if both of them have the same value or not.

Hive Program 2. Average comparing

code

Worksheet *

```
1 CREATE VIEW IF NOT EXISTS ratings_filter AS
2 SELECT movieid, count(ratings) AS number, avg(ratings) AS avghive
3 FROM ratings
4 GROUP BY movieid
5 ORDER BY avghive DESC;
6
7 SELECT movieid , a.title , avghive , a.avg
8 FROM ratings_filter r JOIN alltable a ON r.movieid = a.id;
```

code:

First part, creating a view like the program 1, but this time average ratings is calculated by avg( ) built-in function of hive, and save it as name avghive in ratings_filter. Rest of the code is like hive-program 1.

In second part,select movieid from ratings_filter, a.title(movie names) from alltable, avghive and avg calculated from program 4 (map-reduce) from alltable.

result

movieid	a.title	avghive	a.avg
1653	Entertaining Angels: The Dorothy Day Story (1996)	5.0	5.0
1293	Star Kid (1997)	5.0	5.0
1467	Saint of Fort Washington, The (1993)	5.0	5.0
814	Great Day in Harlem, A (1994)	5.0	5.0
1500	Santa with Muscles (1996)	5.0	5.0
1201	Marlene Dietrich: Shadow and Light (1996)	5.0	5.0
1122	They Made Me a Criminal (1939)	5.0	5.0
1189	Prefontaine (1997)	5.0	5.0
1599	Someone Else's America (1995)	5.0	5.0
1536	Aiqing wansui (1994)	5.0	5.0
1449	Pather Panchali (1955)	4.625	4.625
1594	Everest (1998)	4.5	4.5
119	Maya Lin: A Strong Clear Vision (1994)	4.5	4.5
1398	Anna (1996)	4.5	4.5
1642	Some Mother's Son (1996)	4.5	4.5
408	Close Shave, A (1995)	4.491071428571429	4.49107142857
318	Schindler's List (1993)	4.466442953020135	4.46644295302

A sample of result:

As shown in result, avghive and avg have the same value which it shows both method, using map-reduce and hive(map-reduce feature) work fine.

Therefore, it is an evidence that our result trustable so far.

Hive Program 3

Comparing average

Tables:

1. Alltable

Result**:**

Selected query

Goals

Finding Top-rated movies that at least more than 20 people voted and average rating is more than 4

Hive Program 3. Best movies(Top-rated)

code:

After joinng three table togheter(alltable), it can be seen how using hive makes our job easier.

Select id, title, number(the number of votes each movie has) and avg from all table. By using “WHERE”, a condition can be set. Here we are looking for movies which they average rate is above the 4(avg >4) and at least more than 20 people ranked them.(number >20)

The query is repeated again with adding new condition to our query, where the action column which shows the Action genre equal to 1 (means this movie categorized under Action).

Code without Action

Worksheet

```
1 SELECT id , title , number , avg
2 FROM alltable
3 WHERE avg>4 AND number>20
4 ORDER BY avg DESC;
```

Code with Action

Worksheet

```
1 SELECT id , title , number , avg
2 FROM alltable
3 WHERE avg>4 AND number>20 AND action=1
4 ORDER BY avg DESC;
```

Result without Action

id	title	number	avg
408	Close Shave, A (1995)	112	4.49107142857
318	Schindler's List (1993)	298	4.46644295302
169	Wrong Trousers, The (1993)	118	4.46610169492
483	Casablanca (1942)	243	4.45679012346
114	Wallace & Gromit: The Best of Aardman Animation (1996)	67	4.44776119403
64	Shawshank Redemption, The (1994)	283	4.44522968198
603	Rear Window (1954)	209	4.38755980861
12	Usual Suspects, The (1995)	267	4.38576779026
50	Star Wars (1977)	583	4.35849056604
178	12 Angry Men (1957)	125	4.344
513	Third Man, The (1949)	72	4.33333333333
134	Citizen Kane (1941)	198	4.29292929293
427	To Kill a Mockingbird (1962)	219	4.29223744292

Result with Action

id	title	number	avg
50	Star Wars (1977)	583	4.35849056604
127	Godfather, The (1972)	413	4.28329297821
174	Raiders of the Lost Ark (1981)	420	4.25238095238
313	Titanic (1997)	350	4.24571428571
172	Empire Strikes Back, The (1980)	367	4.20435967302
515	Boot, Das (1981)	201	4.2039800995
187	Godfather: Part II, The (1974)	209	4.18660287081
498	African Queen, The (1951)	152	4.18421052632
173	Princess Bride, The (1987)	324	4.17283950617
22	Braveheart (1995)	297	4.15151515152

Hive Program 4. Worst movies(low-rated)

code:

The only difference between Hive program 4 and Hive program 3 is the in the condition. This time we are looking for movies which they ratings is less than 2 (avg < 2) .

In second part, worst Action movies can be seen.

Let's do something interesting and check the rate of some of these movies from ranking websites like IMDB and Rotten Tomatoes.

Names :mortal kombat annihilation(1997)
IMDB : 3.7/10
Rotten : 2.3/10
Our rate : 1.95/5 number of user : 43

Names :Free Willy 3: The Rescue(1997)
IMDB : 4.7/10
Rotten : 4.9/10
Our rate : 1.74/5 number of user : 27

Names :The Crow: City of Angels(1996)
IMDB : 4.6/10
Rotten : 3.4/10
Our rate : 1.94/5 number of user : 39

Code without Action

Worksheet

```
1 SELECT id , title , number , avg
2 FROM alltable
3 WHERE avg<2 AND number>20
4 ORDER BY avg DESC;
```

Code with Action

Worksheet

```
1 SELECT id , title , number , avg
2 FROM alltable
3 WHERE avg<2 AND number>20 AND action=1
4 ORDER BY avg DESC;
```

Result without Action

id	title	number	avg
375	Showgirls (1995)	23	1.95652173913
890	Mortal Kombat: Annihilation (1997)	43	1.95348837209
743	Crow: City of Angels, The (1996)	39	1.94871794872
1215	Barb Wire (1996)	30	1.93333333333
368	Bio-Dome (1996)	31	1.90322580645
688	Leave It to Beaver (1997)	44	1.84090909091
457	Free Willy 3: The Rescue (1997)	27	1.74074074074
758	Lawnmower Man 2: Beyond Cyberspace (1996)	21	1.71428571429

Result with Action

id	title	number	avg
890	Mortal Kombat: Annihilation (1997)	43	1.95348837209
743	Crow: City of Angels, The (1996)	39	1.94871794872
1215	Barb Wire (1996)	30	1.93333333333

R scatter plots

Comparing average

Tables:

1. **Alltable.sample (100 sample of alltable)**
2. **Using ggplot2 library**

One Sample code**:**

```
> ggplot(all,aes(x=avg,y=alltable.id,color=alltable.id))+geom_point
  (shape=16,size=8,alpha=0.5)+geom_smooth(method = lm,se=FALSE,col
  ="Purple")
```

Goals

Showing the relation between movieIDs and avg and numbers

resources

- StackOverFlow
- Udemy - Taming Big Data with MapReduce and Hadoop - Hands On
- Udemy-Packt.The.Ultimate.Hands.on.Hadoop
- W3Schools.com
- Python learning tutorial - python by mosh Hamedani