SDM ASSIGNMENT 1
(18CS10048)

1.

APPROACH:

First, the respective classes are defined. Then the RDD file is loaded using the parseRatings function that splits the row by "::".

The count of lines is found by using the count scala function on RDD.

Count of unique movies are found by mapping using movie id and then using the distinct count.

Users that gave the most ratings are found by using MapReduce. First used the user id(x) as a key and mapped each entry of rating to x,1. Then reduced them to find the accumulated ratings for each user. Then sorted it in descending order by second argument(that is number of ratings) and output the first entry.

For users with most "5" ratings, I have modified the RDD by filtering out the rows having rating as "5" only and then used the same method as in (d).

OUTPUT:

a. Successfully loaded the rating RDD file
b. 1000209
c. 3706
d. 4169, 2314
e. 4277, 571

```
Spark context Web UI available at http://192.168.1.17:4040
Spark context available as 'sc' (master = local[*], app id = local-1630424093752
).
Spark session available as 'spark'.
(Lines that ratings RDD contain:,1000209)
(Unique movies that have been rated:,3706)
(User who rated the most, number of ratings: ,(4169,2314))
(User who gave the most'5' ratings, number of ratings: ,(4277,571))
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.2
```

CODE:

```scala
import org.apache.spark.rdd.RDD

case class Movie(movie_ID: Integer, title: String, genre: String)
case class User(user_ID: Integer, gender: String, age: Integer, occupation: String, zip_code: String)
case class Rating(user_ID: Integer, movie_ID: Integer, rating: Integer, timestamp: String)

def parseRatings(row: String): Rating = {
    val splitted = row.split("::").map(_.trim).toList
    return Rating(splitted(0).toInt, splitted(1).toInt, splitted(2).toInt, splitted(3))
}

val ratings = sc.textFile("ratings.dat").map(element => parseRatings(element)).cache

println("Lines that ratings RDD contain:",ratings.count)
println("Unique movies that have been rated:",ratings.map(x => x.movie_ID).distinct.count())
println("User who rated the most, number of ratings: ", ratings.keyBy(x => x.user_ID).mapValues(x => 1).reduceByKey((x,y) => x +
  y).sortBy(_._2, false).first)

println("User who gave the most'5' ratings, number of ratings: ",ratings.filter(x => x.rating == 5).keyBy(x => x.user_ID).aggregateByKey(0)
  ((acc, x) => acc + 1, (x, y) => x + y).sortBy(_._2, false).first)
```

2.

APPROACH:

First, the respective classes are defined. Then the RDD file is loaded using the parseRatings, parseMovies, parseUsers function that splits the row by "::".

The number of records in movies and users RDD is calculated by count function scala function on RDD.

Number of comedy movies is found by filtering the movies RDD by its genre with the help of filter and contain function and then the number of comedy movies is found by using the count function.

To find the comedy movie with most ratings, we first filter out the comedy movies as in (b). Then I used the movie id as a key and join this modified table with the ratings RDD. Then I map each row by the movie title and group them which is then mapped to movietitle, number of ratings. Then using the sort in desc, I get the movie with the most ratings.

To find the number of unique users that rated the movies with movie_IDs 2858, 356 and 2329, I first filter the ratings RDD to contain only movies having respective movie_IDs and then group it by user_ID. Then reduced it by the distinct keys clause which remove duplicate rows having same user id. Then using the count function, I have reported the answer.

Created an inverted index on ratings using field movie_ID and printed the first entry.

Created respective movie_ID arrays and joined it with the inverted index. Then mapped using the user ID and found the distinct count.

OUTPUT:

a. Loaded the movies and user RDD successfully. Number of records in movies RDD = 3883, in users RDD = 6040
b. Number of comedy movies = 1200
c. Comedy movie having most ratings = American Beauty (1999),3428
d. Number of unique users without inverted index = 4213
   Time taken = 0.617s
e. Inverted index. First entry =
f. Number of unique users with inverted index = 4213
   Time taken = 9.569s

```
Spark context available as 'sc' (master = local[*], app id = local-1630428084675
).
Spark session available as 'spark'.
(Lines that movies RDD contain:,3883)
(Lines that users RDD contain:,6040)
(Number of comedy movies: ,1200)
(Comedy with most ratings (Movie, Number of rating): ,(American Beauty (1999),34
28))
(Answer to e:,4213)
(Time taken: ,0.617)
First item of inverted index on ratings:
ArrayBuffer(CompactBuffer(Rating(1,1,5,978824268), Rating(6,1,4,978237008), Rati
ng(8,1,4,978233496), Rating(9,1,5,978225952), Rating(10,1,5,978226474), Rating(1
8,1,4,978154768), Rating(19,1,5,978555994), Rating(21,1,3,978139347), Rating(23,
1,4,978463614), Rating(26,1,3,978130703), Rating(28,1,3,978985309), Rating(34,1,
5,978102970), Rating(36,1,5,978061285), Rating(38,1,5,978046225), Rating(44,1,5,
978019369), Rating(45,1,4,977990044), Rating(48,1,4,977975909), Rating(49,1,5,97
7972501), Rating(51,1,5,977947828), Rating(56,1,5,977938855), Rating(60,1,4,9779
31983), Rating(65,1,5,991368774), Rating(68,1,3,991376026), Rating(73,1,3,977867
812), Rating(75,1,5,977851099), Rating(76,1,5,977847069), Rating(78,1,4,97857064
8), Rating(80,1,3,977786904), Rating(90,1,3,993872933), Rating(92,1,4,977646817)
), Rating(5996,1,5,959798556), Rating(6000,1,3,956879000), Rating(6006,1,2,95703
2777), Rating(6010,1,5,957463288), Rating(6011,1,5,956786106), Rating(6013,1,5,9
59116383), Rating(6015,1,5,956778765), Rating(6016,1,4,956778750), Rating(6021,1
,3,956757147), Rating(6022,1,5,956755763), Rating(6025,1,5,956812867), Rating(60
(Answer to g: ,4213)
(Time taken:,9.569)
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.2
```

CODE:

```
import org.apache.spark.rdd.RDD

case class Movie(movie_ID: Integer, title: String, genre: String)
case class User(user_ID: Integer, gender: String, age: Integer, occupation: String, zip_code: String)
case class Rating(user_ID: Integer, movie_ID: Integer, rating: Integer, timestamp: String)


def parseRatings(row: String): Rating = {
    val splitted = row.split("::").map(_.trim).toList
    return Rating(splitted(0).toInt, splitted(1).toInt, splitted(2).toInt, splitted(3))
}

val ratings = sc.textFile("ratings.dat").map(element => parseRatings(element)).cache

def parseMovies(row: String): Movie = {
    val splitted = row.split("::").map(_.trim).toList
    return Movie(splitted(0).toInt, splitted(1), splitted(2))
}

val movies = sc.textFile("movies.dat").map(element => parseMovies(element)).cache
println("Lines that movies RDD contain:", movies.count)



def parseUsers(row: String): User = {
    val splitted = row.split("::").map(_.trim).toList
    return User(splitted(0).toInt, splitted(1), splitted(2).toInt, splitted(3), splitted(4))
}

val users = sc.textFile("users.dat").map(element => parseUsers(element)).cache

println("Lines that users RDD contain:", users.count)


println("Number of comedy movies: ",movies.filter(x => x.genre.contains("Comedy")).count)
```

```
println("Comedy with most ratings (Movie, Number of rating): ", movies.filter(x => x.genre.contains("Comedy")).keyBy(x =>
x.movie_ID).join(ratings.keyBy(x => x.movie_ID)).map(x => (x._2._1.title)).groupBy(x => x).map{case (k, v) => (k, v.size)}.sortBy(x =>
x._2, false).first)


val stime: Long = (System.currentTimeMillis)
println("Answer to e:",ratings.filter(x => (List(2858, 356, 2329).contains(x.movie_ID))).groupBy(x => x.user_ID).keys.distinct.count)
println("Time taken: ", ((System.currentTimeMillis) - stime)/1000.0)


val ratingsInv = ratings.groupBy(r => r.movie_ID).cache

println("First item of inverted index on ratings: ")
print(ratingsInv.lookup(1).take(1))

val stime1: Long = (System.currentTimeMillis)
val items = sc.parallelize(List[Integer](2858, 356, 2329)).keyBy(x => x)
println("Answer to g: ", ratingsInv.join(items).flatMap{x => x._2._1}.map(x => x.user_ID).distinct.count)

println("Time taken:", ((System.currentTimeMillis) - stime1)/1000.0)
```

3.

APPROACH:

Loaded the file and split it by “, | --”.

invertIndex function takes a field and uses groupBy to compile all records with the key into an iterable list mapped to the key and returns this rdd.

To get the answer without inverted kety, I filter out the Rdds with download_id “ghtorrent-22” map them to the repo name of the log if it has any. Then using distinct count, I report the answer.

To get the answer with inverted index, I used filter to search key, “ghtorrent-22” on the inverted index we have created over download_id. Since all this data is on a single executor, I collect it and parallelize it to distribute it on many executors. Then I map it to the repo name and use distinct(), count as in (b).

OUTPUT:

a. Created function invertIndex
b. 4577
c. 4577

```
Spark session available as 'spark'.
(Unique repos with download id ghtorrent-22 without using inverted index:,4577)
21/08/31 23:03:57 WARN TaskSetManager: Stage 4 contains a task of very large siz
e (94910 KiB). The maximum recommended task size is 1000 KiB.
(Unique repos with download id ghtorrent-22 using inverted index:,4577)
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/
```

CODE:

```
import java.util.TimeZone
import java.text.SimpleDateFormat

case class schema (debug_level:String,timestamp:java.util.Date,download_id:String,retrieval_stage:String,rest:String)//

TimeZone.setDefault(TimeZone.getTimeZone("GMT"))
var timeFormat=new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssX")

def repo_names(x:String):String={
        try{
                if(x.contains("github.com/repos/")){
                        var words=x.split("github.com/repos/")(1).split("\\?")(0)
                        var path=words.split("/")
                        if(path.size==1){
                                return path(0)
                        }
                        else if(path.size>1){
                                return path(0)+"/"+path(1)
                        }
                }
                else{
                        var words=x.split(" ")
                        var index=words.indexOf("Repo")
                        return words(index+1)
                }
                return null
        }
        catch{
                case e:Exception=>return null
        }
}


def invertIndex(rdd:org.apache.spark.rdd.RDD[schema],key: String) = {rdd.groupBy(r=>key match{
        case "debug_level" => r.debug_level
        case "timestamp" => r.timestamp
        case "download_id" => r.download_id
        case "retrieval_stage" => r.retrieval_stage
})}

var file=sc.textFile("file:///home/sahil/Desktop/sdm/a1/ghtorrent-logs.txt")
var parsedRDD=file.filter(x=>x.size>0).map(line=>line.split(", | -- ",4))
parsedRDD=parsedRDD.map(line=> if(line.size==4)Array(line(0),line(1),line(2))++line(3).split(": ",2) else line)
var myRDD=parsedRDD.map(x=>try{schema(x(0),timeFormat.parse(x(1)),x(2),x(3),x(4))}catch{case
e:Exception=>schema(null,null,null,null,null)})

val solution1=myRDD.filter(x=>x.rest!=null && x.download_id=="ghtorrent-22" && (x.rest.contains("Repo") ||
x.rest.contains("repos"))).map(x=>repo_names(x.rest)).filter(x=>x!=null).distinct().count()
println("Unique repos with download id ghtorrent-22 without using inverted index:",solution1)



val invert_index=invertIndex(myRDD,"download_id")
val list_log=invert_index.filter(x=>x._1=="ghtorrent-22").flatMap(x=>x._2).collect()
val solution2=sc.parallelize(list_log).filter(x=>(x.rest.contains("Repo") ||
x.rest.contains("repos"))).map(x=>repo_names(x.rest)).filter(x=>x!=null).distinct().count()
println("Unique repos with download id ghtorrent-22 using inverted index:",solution2)
```