Name : Sumit Kumar Yadav Roll No.: 18CS30042

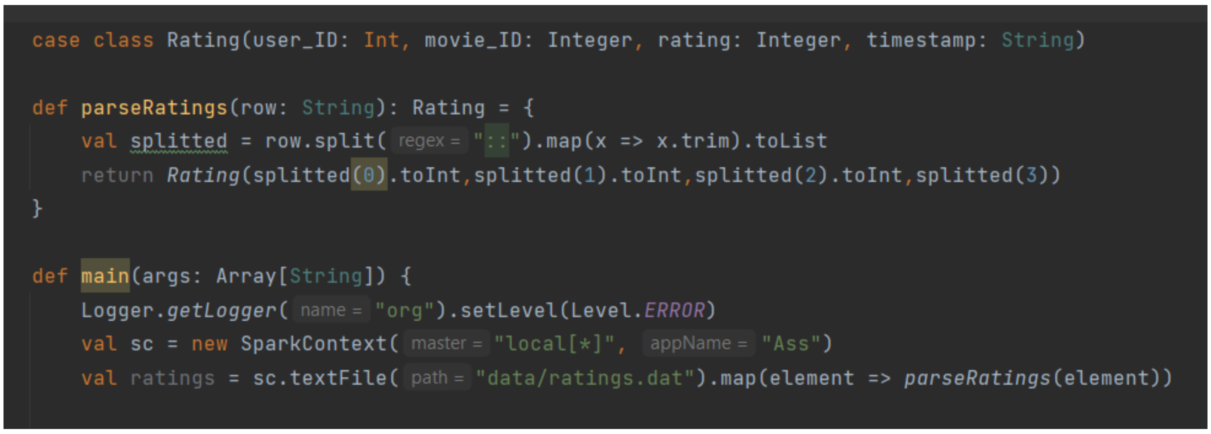
Question 1:

a. Download the ratings file, parse it and load it in an RDD named ratings.

1. At first, I created a case class Rating to store the data in the form of RDD and also assigning the data type of the variables in the dataset.

2. Created a RDD name ratings which parse the whole data of Rating dataset with the help of *parseRatings* function.

3. *parseRatings* function implemented in such a way that take a row from the data file and mapped from ratings.dat to Rating class object and returns an object of Rating.



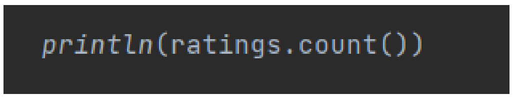
```
case class Rating(user_ID: Int, movie_ID: Integer, rating: Integer, timestamp: String)

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

def main(args: Array[String]) {
    Logger.getLogger( name = "org").setLevel(Level.ERROR)
    val sc = new SparkContext( master = "local[*]", appName = "Ass")
    val ratings = sc.textFile( path = "data/ratings.dat").map(element => parseRatings(element))
```

b. How many lines does the ratings RDD contain?

1. With the help of count() method,I calculated the number of lines in ratings RDD



```
println(ratings.count())
```

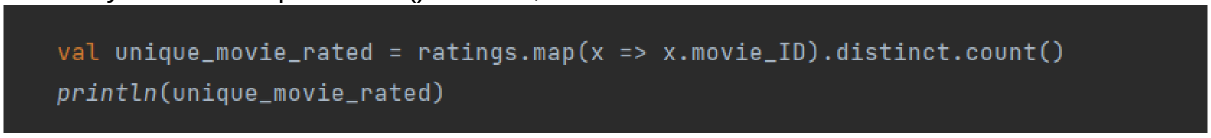
<u>Output:</u> 1000209

c. Count how many unique movies have been rated.

1. Map is used to convert the rating RDD to another RDD that only contains the movie_ID.

2. Now, with the help of distinct() method, I removed all the duplicated entries from the previously created RDD.

3. Finally with the help of count() method, the number of entries are calculated.



```
val unique_movie_rated = ratings.map(x => x.movie_ID).distinct.count()
println(unique_movie_rated)
```

<u>Output:</u> 3706

d. Which user gave the most ratings? Return the userID and number of ratings.

1. Map is used to convert the rating RDD to another RDD that only contains the user_ID.

2. with the help of countByvalue(), the number of entries per user_id are calculated now mapped this as a pair whose first element is no_of_ratings and second element is corresponding user_ID

3. Finally, with the help of max function maximum value is calculated whose having most no of ratings

```
val a = ratings.map(x => x.user_ID).countByValue().map(x => (x._2,x._1)).max
println("User_ID : " + a._2 + ", no_of_ratings : " +a._1)
```

<u>Output:</u> User_ID : 4169, no_of_ratings : 2314

e. Which user gave the most '5' ratings? Return the userID and number of ratings.

1. Similar to the previous case I calculated the most rating but initially with the help of filter function I filter out the ratings RDD whose rating is 5

2. here also countByvalue() is used to count the number of entries per user_id

3. At last, I will take the maximum value corresponding to the most 5 ratings

```
val a= ratings.filter(x => x.rating == 5).map(x => x.user_ID).countByValue().map(x => (x._2,x._1)).max
println("User_ID : " + a._2 + ", no_of_ratings : " +a._1)
```

<u>Output:</u> User_ID : 4277, no_of_ratings : 571

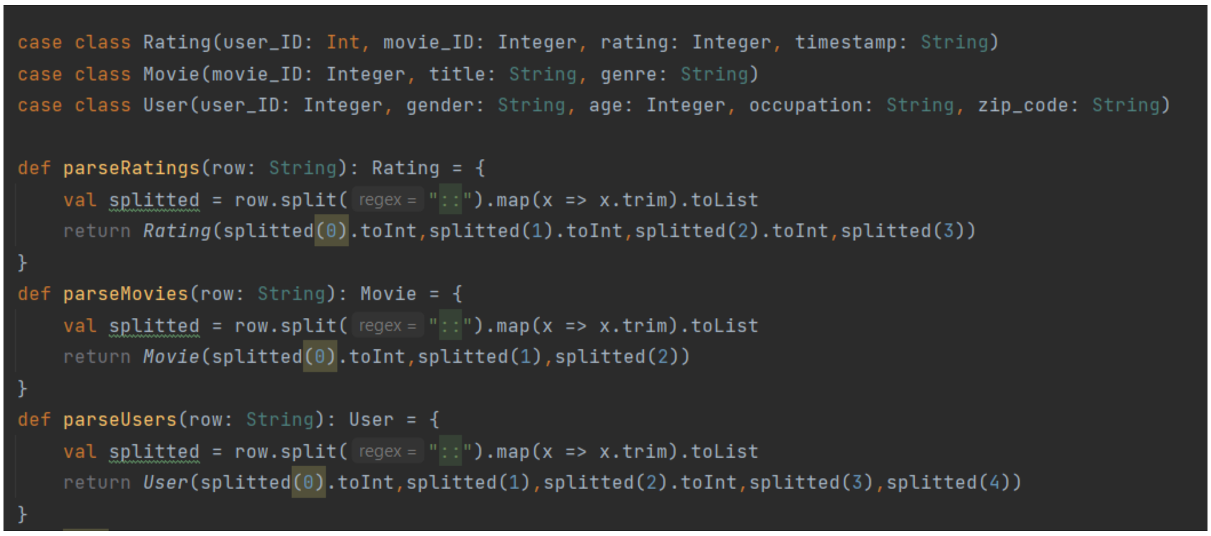
<u>Question 2:</u>

a. Read the movies and users files into RDDs. How many records are there in each RDD?

1. At first, I created a case class Movie and case class User to store the data in the form of RDD and also assigning the data type of the variables in the dataset.

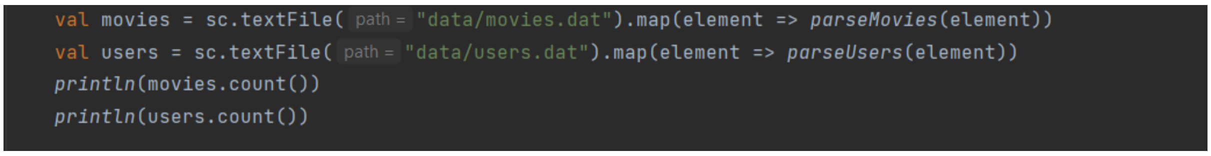
2. Created a RDD name movies and users which parse the whole data of Movie and User dataset with the help of *parseMovies* and *parseUsers* function respectively.

3. *parseMovies* and *parseUsers* functions are implemented in such a way that take a row from the data file and mapped from movies.dat and users.dat to Movie and User class object respectively and returns the corresponding object.



```
case class Rating(user_ID: Int, movie_ID: Integer, rating: Integer, timestamp: String)
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)

def parseRatings(row: String): Rating = {
    val splitted = row.split( regex = "::").map(x => x.trim).toList
    return Rating(splitted(0).toInt,splitted(1).toInt,splitted(2).toInt,splitted(3))
}
def parseMovies(row: String): Movie = {
    val splitted = row.split( regex = "::").map(x => x.trim).toList
    return Movie(splitted(0).toInt,splitted(1),splitted(2))
}
def parseUsers(row: String): User = {
    val splitted = row.split( regex = "::").map(x => x.trim).toList
    return User(splitted(0).toInt,splitted(1),splitted(2).toInt,splitted(3),splitted(4))
}
```



```
    val movies = sc.textFile( path = "data/movies.dat").map(element => parseMovies(element))
    val users = sc.textFile( path = "data/users.dat").map(element => parseUsers(element))
    println(movies.count())
    println(users.count())
```

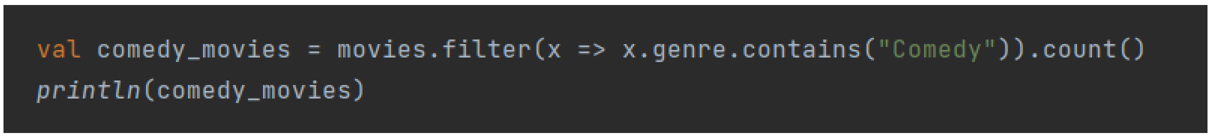
<u>Output:</u>

Movies.count(): 3883

User.count(): 6040

b. How many of the movies are a comedy?

1. With the help of filter method, Filtered the movies that contains “Comedy” genre in the movies RDD.



```
val comedy_movies = movies.filter(x => x.genre.contains("Comedy")).count()
println(comedy_movies)
```

<u>Output:</u> 1200

c. Which comedy has the most ratings? Return the title and the number of rankings. Answer this question by joining two datasets.

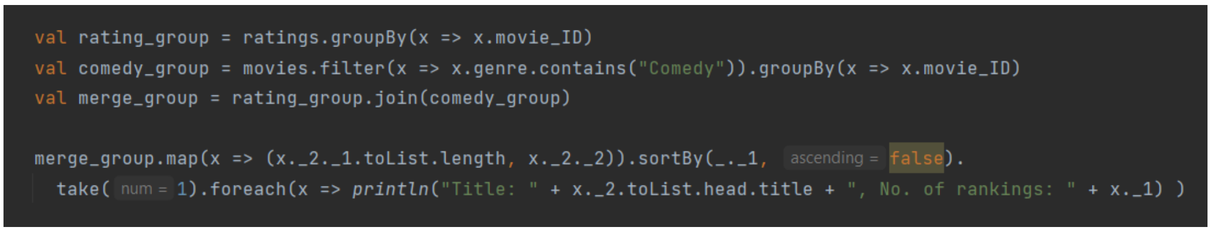
1. At first, I created the inverted index rating_group from rating RDD using groupby function

2. Similarly for the movie RDD, created the inverted index comedy_group from movie RDD using groupby function

3. Now, Join the 2 group that are cretaed earlier ie, rating_group and comedy_group using join method and name as merge_group

4. then merged the merge_group RDD as (movie_ID, (rating list, movie))

5. After this I map this RDD to (rating list.length, movie) and sort it in descending order of first element and print the first element in this sorted list



```
val rating_group = ratings.groupBy(x => x.movie_ID)
val comedy_group = movies.filter(x => x.genre.contains("Comedy")).groupBy(x => x.movie_ID)
val merge_group = rating_group.join(comedy_group)

merge_group.map(x => (x._2._1.toList.length, x._2._2)).sortBy(_._1, ascending = false).
  take( num = 1).foreach(x => println("Title: " + x._2.toList.head.title + ", No. of rankings: " + x._1) )
```

<u>Output:</u>

Title: American Beauty (1999), No. of rankings: 3428

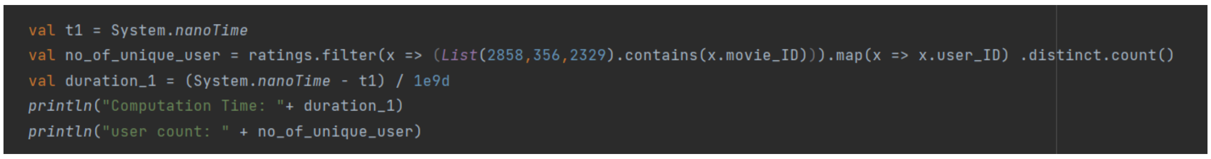
e. Compute the number of unique users that rated the movies with movie_IDs 2858, 356 and 2329 without using an inverted index. Measure the time (in seconds) it takes to make this computation.

1. With the help of filter function, filtered all the ratings whose movie Id = 2858 or 356 or 2329

2. Now from these filtered ratings to RDD that contains just user_id with the help of map and after this used distinct() to remove the duplicates entries

3. Finally using count() function, calculated the no_of_unique_user

4. Apart from this, The computation time is also calculated



```
val t1 = System.nanoTime
val no_of_unique_user = ratings.filter(x => (List(2858,356,2329).contains(x.movie_ID))).map(x => x.user_ID) .distinct.count()
val duration_1 = (System.nanoTime - t1) / 1e9d
println("Computation Time: "+ duration_1)
println("user count: " + no_of_unique_user)
```

<u>Output:</u>

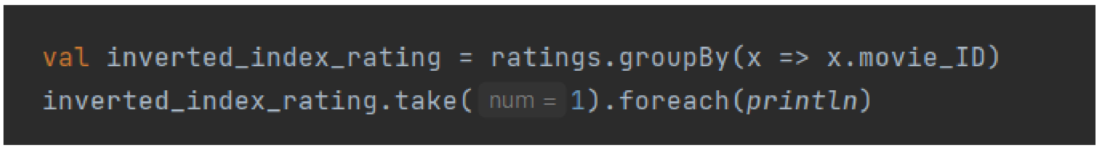
Computation Time: 2.7582553

user count: 4213

f. Create an inverted index on ratings, field movie_ID. Print the first item.

1. created the inverted index using groupby method.

2. With the help of take function, printed the first iem of the inverted index ratings.



```
val inverted_index_rating = ratings.groupBy(x => x.movie_ID)
inverted_index_rating.take( num = 1).foreach(println)
```

<u>Output:</u>

(3586,CompactBuffer(Rating(238,3586,4,976766809), Rating(1059,3586,1,974951211), Rating(1321,3586,1,974778231), Rating(1448,3586,4,977257642), Rating(1590,3586,5,976225469), Rating(2124,3586,4,974653584), Rating(2290,3586,2,974522135), Rating(2402,3586,4,974263741), Rating(2860,3586,3,974232584), Rating(3618,3586,2,967119492),

Rating(3841,3586,3,965999655), Rating(4044,3586,4,965499419),
Rating(4050,3586,3,966620842), Rating(4054,3586,4,965510798),
Rating(4169,3586,3,971580786), Rating(4207,3586,4,965324249),
Rating(4408,3586,4,965168003), Rating(4510,3586,2,965930044),
Rating(4543,3586,5,964670575), Rating(4754,3586,4,963186331),
Rating(5068,3586,4,962469697), Rating(5100,3586,4,962739858),
Rating(5111,3586,4,962334345), Rating(5156,3586,5,1042222175),
Rating(5262,3586,5,961194303), Rating(5393,3586,2,960322348),
Rating(5426,3586,2,960100002), Rating(5795,3586,3,958062476)))

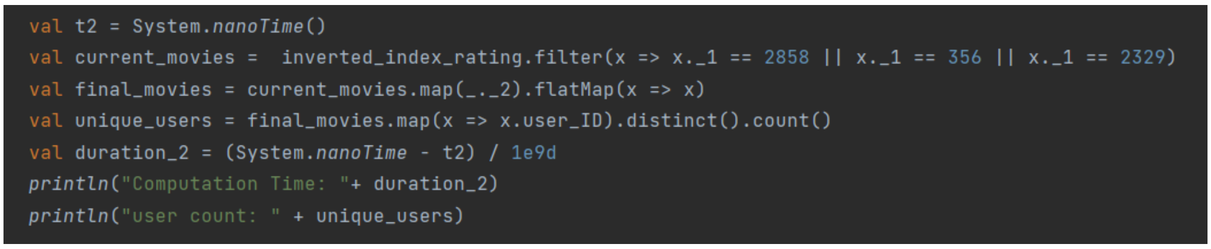
g. Compute the number of unique users that rated the movies with movie_IDs 2858, 356 and 2329 using the above calculated index. Measure the time (in seconds) it takes to compute the same result using the index.

1. At first, filtered the inverted index with objects whose having movie Id = 2858 or 356 or 2329

2. Then transformed with a map containing only the second element with the help of flatmap that further transformed the RDD so that it only contains userID and used distinct to remove similar entries

3. Finally using count() function, calculated the no_of_unique_user

4. Apart from this, The computation time is also calculated



```
val t2 = System.nanoTime()
val current_movies =  inverted_index_rating.filter(x => x._1 == 2858 || x._1 == 356 || x._1 == 2329)
val final_movies = current_movies.map(_._2).flatMap(x => x)
val unique_users = final_movies.map(x => x.user_ID).distinct().count()
val duration_2 = (System.nanoTime - t2) / 1e9d
println("Computation Time: "+ duration_2)
println("user count: " + unique_users)
```

<u>Output:</u>

Computation Time: 1.3605593

user count: 4213

<u>Question 3</u>

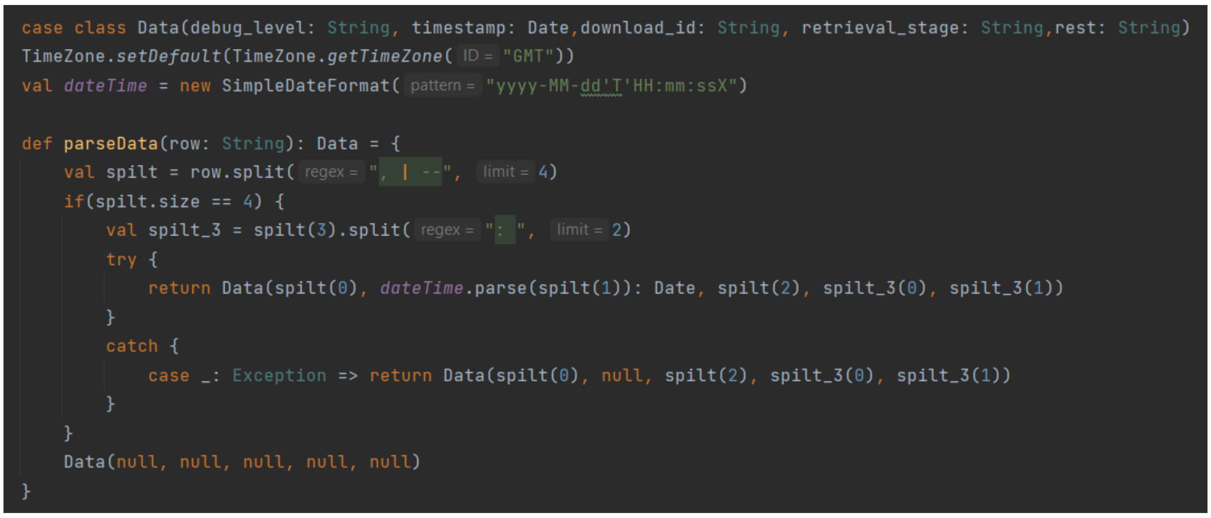
a. Create a function that given an RDD and a field (e.g. download_id), it computes an inverted index on the RDD for efficiently searching the records of the RDD using values of the field as keys.

1. At first, I created a case class Data to store the data in the form of RDD and also assigning the data type of the variables in the dataset.

2. Created a RDD name Assignment_1 which parse the whole data of dataset with the help of *parseData* function.

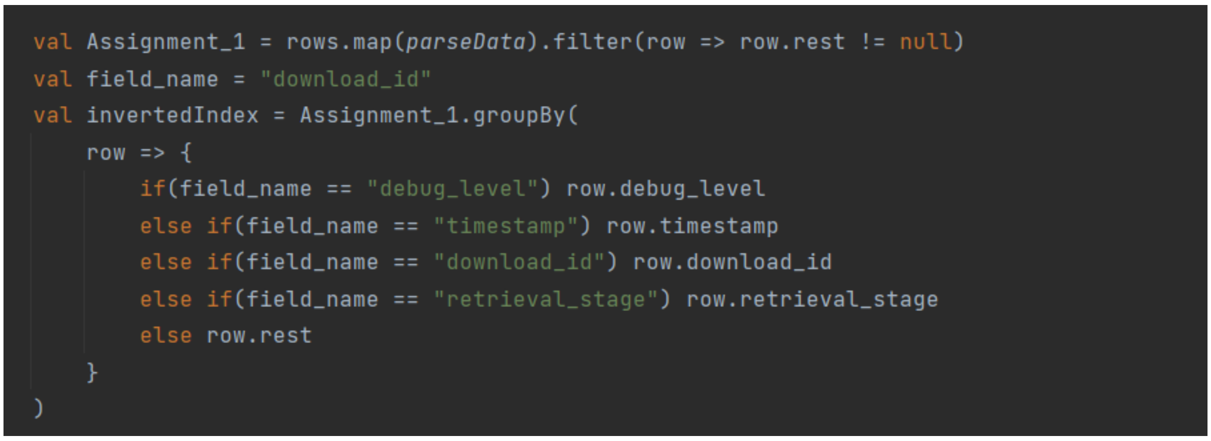
3. Finally, Created the inverted_index using groupby() method

4. field_name is used to update as per requirement for inverted index



```
case class Data(debug_level: String, timestamp: Date,download_id: String, retrieval_stage: String,rest: String)
TimeZone.setDefault(TimeZone.getTimeZone( ID = "GMT"))
val dateTime = new SimpleDateFormat( pattern = "yyyy-MM-dd'T'HH:mm:ssX")

def parseData(row: String): Data = {
    val spilt = row.split( regex = ", | --",  limit = 4)
    if(spilt.size == 4) {
        val spilt_3 = spilt(3).split( regex = ": ",  limit = 2)
        try {
            return Data(spilt(0), dateTime.parse(spilt(1)): Date, spilt(2), spilt_3(0), spilt_3(1))
        }
        catch {
            case _: Exception => return Data(spilt(0), null, spilt(2), spilt_3(0), spilt_3(1))
        }
    }
    Data(null, null, null, null, null)
}
```

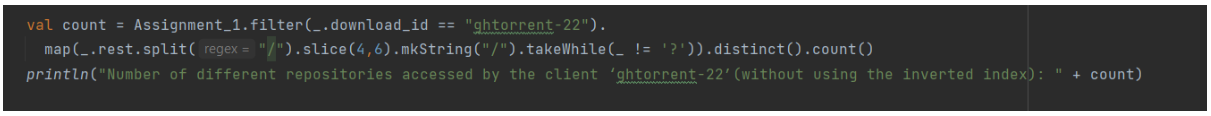


```
val Assignment_1 = rows.map(parseData).filter(row => row.rest != null)
val field_name = "download_id"
val invertedIndex = Assignment_1.groupBy(
    row => {
        if(field_name == "debug_level") row.debug_level
        else if(field_name == "timestamp") row.timestamp
        else if(field_name == "download_id") row.download_id
        else if(field_name == "retrieval_stage") row.retrieval_stage
        else row.rest
    }
)
```

b. Compute the number of different repositories accessed by the client 'ghtorrent-22' (without using the inverted index).

1. At firstly, filtered all the objects from the created RDD whose having download_id = ghtorrent-22

2. Then extracted the repo name by making the slice with '/' and taking the 4th element to the 6th element and making to again string by contracting it with '/'



```
val count = Assignment_1.filter(_.download_id == "ghtorrent-22").
  map(_.rest.split( regex = "/").slice(4,6).mkString("/").takeWhile(_ != '?')).distinct().count()
println("Number of different repositories accessed by the client 'ghtorrent-22'(without using the inverted index): " + count)
```

Output:

Number of different repositories accessed by the client 'ghtorrent-22'(without using the inverted index): 3973

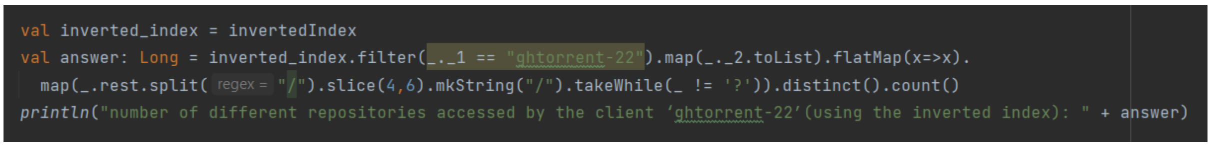
c. Compute the number of different repositories accessed by the client 'ghtorrent-22' using the inverted index calculated above.

1. Firstly created an inverted index using groupby method

2. Then filter the inverted index using filter method whose having download_id= ghtorrent-22

And then transform it using map to a RDD containing an only second element of inverted index objects.

3. Finally, split similar to as done in part(b) and use count function to calculate the distinct repo available



```scala
val inverted_index = invertedIndex
val answer: Long = inverted_index.filter(_._1 == "ghtorrent-22").map(_._2.toList).flatMap(x=>x).
  map(_.rest.split( regex = "/").slice(4,6).mkString("/").takeWhile(_ != '?')).distinct().count()
println("number of different repositories accessed by the client ‘ghtorrent-22’(using the inverted index): " + answer)
```

<u>**Output:**</u>

number of different repositories accessed by the client ‘ghtorrent-22’(using the inverted index): 3973