IMPORTING LIBRARIES:

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext.implicits._
import org.apache.spark.sql._
import org.apache.spark.mllib.recommendation.{ALS, MatrixFactorizationModel, Rating}
```

DEFINING THE SCHEMAS USING A CASE CLASSES: (input format MovieID::Title::Genres)

```
case class Movie(movieId: Int, title: String)
case class User(userId: Int, gender: String, age: Int, occupation: Int, zip: String)
```

Function to parse input into Movie class

```
def parseMovie(str: String): Movie = {
 val fields = str.split("::")
 assert(fields.size == 3)
 Movie(fields(0).toInt, fields(1))
}
```

Function to parse input into User class

```
def parseUser(str: String): User = {
 val fields = str.split("::")
 assert(fields.size == 5)
 User(fields(0).toInt, fields(1).toString, fields(2).toInt, fields(3).toInt, fields(4).toString)
}
```

Function to parse input UserID::MovieID::Rating

and pass into constructor for org.apache.spark.mllib.recommendation.Rating class

```
def parseRating(str: String): Rating = {
 val fields = str.split("::")
 Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble)
}
```

LOADING THE DATA INTO RDD:

val ratingText = sc.textFile("C:/Users/ramgo/Desktop/project#1/ratings.dat")

val ratingsRDD = ratingText.map(parseRating).cache()

SUMMARY STATISTICS:

Counting the total number of ratings:

val numRatings = ratingsRDD.count()

Counting number of users who rated a movie

val numUsers = ratingsRDD.map(_.user).distinct().count()

Counting number of movies rated

val numMovies = ratingsRDD.map(_.product).distinct().count()

```
scala> val numRatings = ratingsRDD.count()
numRatings: Long = 1000209

scala> val numUsers = ratingsRDD.map(_.user).distinct().count()
numUsers: Long = 6040

scala> val numMovies = ratingsRDD.map(_.product).distinct().count()
numMovies: Long = 3706
```

LOADING THE DATA INTO DATAFRAMES:

val moviesDF= sc.textFile("C:/Users/ramgo/Desktop/project#1/movies.dat").map(parseMovie).toDF()

val usersDF = sc.textFile("C:/Users/ramgo/Desktop/project#1/users.dat").map(parseUser).toDF()

CREATING A DATAFRAME FROM RATINGS RDD

val ratingsDF = ratingsRDD.toDF()

DISPLAYING THE DATA FRAME

moviesDF.take(5).foreach(println)

```
scala> moviesDF.take(5).foreach(println)
[1,Toy Story (1995)]
[2,Jumanji (1995)]
[3,Grumpier Old Men (1995)]
[4,Waiting to Exhale (1995)]
[5,Father of the Bride Part II (1995)]
```

CONVERTING THEM INTO SQL TABLES

ratingsDF.registerTempTable("ratings")

moviesDF.registerTempTable("movies")

usersDF.registerTempTable("users")

SUMMARY STATISTICS:

ratingsDF.select("product").distinct.count

```
scala> ratingsDF.select("product").distinct.count
res4: Long = 3706
```

ratingsDF.groupBy("product", "rating").count.show

```
scala> ratingsDF.groupBy("product", "rating").count.show
+-------+------+-----+
|product|rating|count|
+-------+------+-----+
|    457|   5.0|  645|
|   2908|   3.0|  153|
|   3948|   3.0|  222|
|   2539|   4.0|  232|
|    517|   3.0|  157|
|     34|   5.0|  563|
|    496|   3.0|   10|
|    261|   5.0|   50|
|   1676|   3.0|  333|
|   3052|   5.0|  272|
|   3152|   5.0|  110|
|   3268|   1.0|   98|
|   3097|   4.0|  101|
|   2269|   5.0|    2|
|    419|   2.0|   54|
|   1047|   3.0|  121|
|   1499|   2.0|  123|
|   3088|   3.0|   64|
|   1731|   1.0|   32|
|   2899|   1.0|    4|
+-------+------+-----+
```

ratingsDF.groupBy("product").count.agg(min("count"), avg("count"),max("count")).show

```
scala> ratingsDF.groupBy("product").count.agg(min("count"), avg("count"),max("count")).show
+----------+------------------+----------+
|min(count)|        avg(count)|max(count)|
+----------+------------------+----------+
|         1|269.88909875876953|      3428|
+----------+------------------+----------+
```

ratingsDF.select("product", "rating").groupBy("product", "rating").count.agg(min("count"), avg("count"),max("count")).show

```
+----------+------------------+----------+
|min(count)|        avg(count)|max(count)|
+----------+------------------+----------+
|         1|59.141970198675494|      1963|
+----------+------------------+----------+
```

COUNTING THE MAXIMUM AND MINIMUM RATINGS AND THE NUMBER OF USERS WHO RATED THE MOVIE.

val results =sqlContext.sql("select movies.title, movierates.maxr, movierates.minr, movierates.cntu from(SELECT ratings.product, max(ratings.rating) as maxr, min(ratings.rating) as minr,count(distinct user) as cntu FROM ratings group by ratings.product) movierates join movies on movierates.product=movies.movieId order by movierates.cntu desc ")

results.take(20).foreach(println)

```
scala> results.take(20).foreach(println)
[American Beauty (1999),5.0,1.0,3428]
[Star Wars: Episode IV - A New Hope (1977),5.0,1.0,2991]
[Star Wars: Episode V - The Empire Strikes Back (1980),5.0,1.0,2990]
[Star Wars: Episode VI - Return of the Jedi (1983),5.0,1.0,2883]
[Jurassic Park (1993),5.0,1.0,2672]
[Saving Private Ryan (1998),5.0,1.0,2653]
[Terminator 2: Judgment Day (1991),5.0,1.0,2649]
[Matrix, The (1999),5.0,1.0,2590]
[Back to the Future (1985),5.0,1.0,2583]
[Silence of the Lambs, The (1991),5.0,1.0,2578]
[Men in Black (1997),5.0,1.0,2538]
[Raiders of the Lost Ark (1981),5.0,1.0,2514]
[Fargo (1996),5.0,1.0,2513]
[Sixth Sense, The (1999),5.0,1.0,2459]
[Braveheart (1995),5.0,1.0,2443]
[Shakespeare in Love (1998),5.0,1.0,2369]
[Princess Bride, The (1987),5.0,1.0,2318]
[Schindler's List (1993),5.0,1.0,2304]
[L.A. Confidential (1997),5.0,1.0,2288]
[Groundhog Day (1993),5.0,1.0,2278]
```

DISPLAYING THE TOP 10 ACTIVE USERS AND THE NUMBER OF TIMES THEY HAVE RATED A MOVIE:

val mostActiveUsersSchemaRDD = sqlContext.sql("SELECT ratings.user, count(*) as ct from ratings group by ratings.user order by ct desc limit 10")

mostActiveUsersSchemaRDD.take(20).foreach(println)

```
scala> mostActiveUsersSchemaRDD.take(20).foreach(println)
[4169,2314]
[1680,1850]
[4277,1743]
[1941,1595]
[1181,1521]
[889,1518]
[3618,1344]
[2063,1323]
[1150,1302]
[1015,1286]
```

RANDOMLY SPLITTING THE RATINGS RDD INTO TRAINING DATA RDD (80%) AND TEST DATA RDD (20%)

val splits = ratingsRDD.randomSplit(Array(0.8, 0.2), 0L)

val trainingRatingsRDD = splits(0).cache()

val testRatingsRDD = splits(1).cache()

val numTraining = trainingRatingsRDD.count()

val numTest = testRatingsRDD.count()

```
scala> val numTraining = trainingRatingsRDD.count()
numTraining: Long = 799402

scala> val numTest = testRatingsRDD.count()
numTest: Long = 200807
```

BUILDING THE RECOMMENDATION MODEL USING ALS WITH RANK=20, ITERATIONS=10

val model = ALS.train(trainingRatingsRDD, 20, 10)

val model = (new ALS().setRank(20).setIterations(10).run(trainingRatingsRDD))

MAKING MOVIE PREDICTIONS FOR USER 1181

val topRecsForUser = model.recommendProducts(1181, 10)

```
scala> topRecsForUser.take(5).foreach(println)
Rating(1181,1859,5.0290287281228485)
Rating(1181,2323,4.70397994448498)
Rating(1181,1889,4.270146634362074)
Rating(1181,3188,4.268032784078663)
Rating(1181,1218,4.234743719244974)
```

1859::Taste of Cherry (1997)
2323::Cruise, The (1998)
1889::Insomnia (1997)
3188::Life and Times of Hank Greenberg, The (1998)
1218::Killer, The (Die xue shuang xiong) (1989)