Get SparkContext

```
In [1]: from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName("recsys").getOrCreate()
        sc = spark.sparkContext
```

```
In [2]: from pyspark.sql.types import Row
```

Loading Artist-name data

```
In [3]: rawArtistData = sc.textFile("../Dropbox/pj_ss/audio/artist_data.txt")
```

```
In [4]: rawArtistData.take(10)
```

```
Out[4]: ['1134999\t06Crazy Life',
         '6821360\tPang Nakarin',
         '10113088\tTerfel, Bartoli- Mozart: Don',
         '10151459\tThe Flaming Sidebur',
         '6826647\tBodenstandig 3000',
         '10186265\tJota Quest e Ivete Sangalo',
         '6828986\tToto_XX (1977',
         '10236364\tU.S Bombs -',
         '1135000\tartist formaly know as Mat',
         '10299728\tKassierer - Musik für beide Ohren']
```

```
In [5]: artistDataDF = rawArtistData\
            .map(lambda line: line.split("\t", 1))\
            .filter(lambda x: len(x) == 2)\
            .filter(lambda x: x[0].isdigit())\
            .map(lambda x: Row(artist=int(x[0]), name=x[1]))\
            .toDF()
```

```
In [6]: artistDataDF.take(2)
```

```
Out[6]: [Row(artist=1134999, name='06Crazy Life'),
         Row(artist=6821360, name='Pang Nakarin')]
```

Loading Artist alias

```
In [7]: rawArtistAlias = sc.textFile('../Dropbox/pj_ss/audio/artist_alias.txt')
In [8]: rawArtistAlias.take(10)
```

```
Out[8]: ['1092764\t1000311',
         '1095122\t1000557',
         '6708070\t1007267',
         '10088054\t1042317',
         '1195917\t1042317',
         '1112006\t1000557',
         '1187350\t1294511',
         '1116694\t1327092',
         '6793225\t1042317',
         '1079959\t1000557']
```

```
In [9]: artistAliasDF = rawArtistAlias\
            .map(lambda line: line.split("\t"))\
            .filter(lambda x: len(x) == 2)\
            .filter(lambda x: x[0].isdigit())\
            .filter(lambda x: x[1].isdigit())\
            .map(lambda x: Row(artist=int(x[0]), alias=int(x[1])))\
            .toDF()
```

```
In [10]: artistAliasDF.take(2)
```

```
Out[10]: [Row(alias=1000311, artist=1092764), Row(alias=1000557, artist=109512
         2)]
```

Loading User-Artist data

```
In [11]: rawUserArtistData = sc.textFile("../Dropbox/pj_ss/audio/user_artist_dat
         a.txt")
```

```
In [12]: rawUserArtistData.first()
```

```
Out[12]: '1000002 1 55'
```

```
In [14]: userArtistDF = rawUserArtistData\
             .map(lambda line: line.split(" "))\
             .map(lambda x: Row(user=int(x[0]), artist=int(x[1]), count=int(x[2
         ])))\
             .toDF()
```

```
In [15]: userArtistDF.take(2)
```

```
Out[15]: [Row(artist=1, count=55, user=1000002),
          Row(artist=1000006, count=33, user=1000002)]
```

```
In [16]: print(userArtistDF.select('user').rdd.max())
         print(userArtistDF.select('user').rdd.min())
         print(userArtistDF.select('artist').rdd.max())
         print(userArtistDF.select('artist').rdd.min())

         Row(user=2443548)
         Row(user=90)
         Row(artist=10794401)
         Row(artist=1)
```

Resolve Aliasing

```
In [17]: trainData = userArtistDF.join(artistAliasDF, "artist", "left_outer")\
             .rdd\
             .map(lambda x: (x['user'], x['artist'], x['count']) if x['alias']==
         None \
                   else (x['user'], x['alias'], x['count']))\
             .map(lambda x: (int(x[0]), int(x[1]), int(x[2])))
```

```
In [18]: trainData.take(10)

Out[18]: [(1000159, 26, 1),
          (1000320, 26, 1),
          (1000385, 26, 1),
          (1000404, 26, 3),
          (1000405, 26, 9),
          (1000415, 26, 1),
          (1000518, 26, 1),
          (1000573, 26, 63),
          (1000596, 26, 3),
          (1000626, 26, 5)]
```

```
In [19]: trainData.cache()

Out[19]: PythonRDD[69] at RDD at PythonRDD.scala:49
```

Build Model

```
In [20]: from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel,
         Rating

         rank = 10
         iterations = 5
         lbda = 0.01
         seed = 42
```

```
In [21]: #model = ALS.train(trainData, rank=rank, iterations=iterations, lambda_=
         lbda, seed=seed)
         model = ALS.trainImplicit(trainData,
                                   rank=rank,
                                   iterations=iterations,
                                   lambda_=lbda,
                                   alpha=1.0,
                                   seed=seed)
```

```
In [22]: model
```

```
Out[22]: <pyspark.mllib.recommendation.MatrixFactorizationModel at 0x1064c3eb8>
```

Model의 사용

```
In [23]: model.userFeatures().take(1)
```

```
Out[23]: [(7400,
           array('d', [-0.5253109335899353, -3.717756509780884, 0.55670106410980
         22, -0.6934528350830078, 1.8062853813171387, -0.573974609375, 2.7792887
         687683105, 0.9534238576889038, -2.383761405944824, -0.750530898571014
         4]))]
```

```
In [24]: model.productFeatures().take(1)
```

```
Out[24]: [(100,
           array('d', [-0.08007114380598068, -0.13515709340572357, -0.0206525325
         7751465, -0.008643229492008686, 0.03264828398823738, 0.0178102459758520
         13, 0.06439027190208435, 0.06781982630491257, -0.08726055175065994, -0.
         10340531915426254]))]
```

```
In [25]: productId = 1
         model.recommendUsers(productId, 5)
```

```
Out[25]: [Rating(user=1037240, product=1, rating=1.3606984320279356),
          Rating(user=1054417, product=1, rating=1.336792097354692),
          Rating(user=2167160, product=1, rating=1.3278224025483398),
          Rating(user=1005353, product=1, rating=1.3089582300854892),
          Rating(user=1001440, product=1, rating=1.3073439566057818)]
```

```
In [26]: userId = 2010581
         model.recommendProducts(userId, 5)
```

```
Out[26]: [Rating(user=2010581, product=1077309, rating=1.2734482951652937),
          Rating(user=2010581, product=1010524, rating=1.244074256266154),
          Rating(user=2010581, product=1000602, rating=1.2122672161252315),
          Rating(user=2010581, product=1000175, rating=1.2115658940753997),
          Rating(user=2010581, product=1010267, rating=1.198888835210903)]
```

```
In [27]: model.predict(userId, productId)
```

```
Out[27]: 0.5197372067168629
```

In [29]:
```
recommendedArtistIDs = [ x[1] for x in model.recommendProducts(userId, 5
) ]
artistDataDF.filter(artistDataDF.artist.isin(recommendedArtistIDs)).show
()
```

```
+-------+-------------+
| artist|         name|
+-------+-------------+
|1000175|     Chevelle|
|1010267|    Zebrahead|
|1077309| Billy Talent|
|1010524|     MC Chris|
|1000602|Finger Eleven|
+-------+-------------+
```

Calculate MSE

In [30]:
```
testData = trainData.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testData).map(lambda r: ((r[0], r[1]), r[
2]))
```

In [31]:
```
ratesAndPreds = trainData.map(lambda r: ((r[0], r[1]), r[2])).join(predi
ctions)
```

In [32]:
```
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
```

In [33]:
```
print("Mean Squared Error = " + str(MSE))
```

```
Mean Squared Error = 24103.494386597584
```

Save and Load for later use

In [35]:
```
model.save(sc, '../pyspark/recsysmodel')
```

In [36]:
```
sameModel = MatrixFactorizationModel.load(sc, '../pyspark/recsysmodel')
```