#Data Loading from CSV into SPARK RDD

def set\_hadoop\_config\_with\_credentials\_f0a64df105924d5e8517e85b6a0695ce(name):

'''This function sets the Hadoop configuration so it is possible to

access data from Bluemix Object Storage using Spark'''

prefix = 'fs.swift.service.' + name

hconf = sc.\_jsc.hadoopConfiguration()

hconf.set(prefix + '.auth.url', 'https://identity.open.softlayer.com'+'/v3/auth/tokens')

hconf.set(prefix + '.auth.endpoint.prefix', 'endpoints')

hconf.set(prefix + '.tenant', 'c1bc603f45a944c8bebecf4459b886a4')

hconf.set(prefix + '.username', '5c7793bbf03b4d85b3e0458e5930e396')

hconf.set(prefix + '.password', 'UILc^{]BQI0s}U8M')

hconf.setInt(prefix + '.http.port', 8080)

hconf.set(prefix + '.region', 'dallas')

hconf.setBoolean(prefix + '.public', False)

name = 'keystone'

set\_hadoop\_config\_with\_credentials\_f0a64df105924d5e8517e85b6a0695ce(name)

#Spark RDDs of all four CSVs:

Ratings\_Data = sc.textFile("swift://DefaultProjectajayrathore27gmailcom." + name + "/ratings.csv")

#Ratings\_Data.take(5)

Links\_Data = sc.textFile("swift://DefaultProjectajayrathore27gmailcom." + name + "/links.csv")

#Links\_Data.take(5)

Movies\_Data = sc.textFile("swift://DefaultProjectajayrathore27gmailcom." + name + "/movies.csv")

#Movies\_Data.take(5)

Tags\_Data = sc.textFile("swift://DefaultProjectajayrathore27gmailcom." + name + "/tags.csv")

#Tags\_Data.take(5)

Ratings\_Data.take(5)

#Filtering headers to remove for main Data

Ratings\_Data\_header = Ratings\_Data.take(1)[0]

Links\_Data\_header = Links\_Data.take(1)[0]

Movies\_Data\_header = Movies\_Data.take(1)[0]

Tags\_Data\_header = Tags\_Data.take(1)[0]

#Data Filtering

Ratings\_DF = Ratings\_Data.filter(lambda line: line!=Ratings\_Data\_header)\

.map(lambda line: line.split(",")).map(lambda x: (x[0],x[1],x[2])).cache()

#Ratings\_DF.take(3)

Links\_DF = Links\_Data.filter(lambda line: line!=Links\_Data\_header)\

.map(lambda line: line.split(",")).map(lambda x: (x[0],x[1],x[2])).cache()

#Links\_DF.take(3)

Movies\_DF = Movies\_Data.filter(lambda line: line!=Movies\_Data\_header)\

.map(lambda line: line.split(",")).map(lambda x: (x[0],x[1])).cache()

#Movies\_DF.take(3)

Movies\_titles = Movies\_DF.map(lambda x: (int(x[0]),x[1]))

Tags\_DF = Tags\_Data.filter(lambda line: line!=Tags\_Data\_header)\

.map(lambda line: line.split(",")).map(lambda tokens: (tokens[0],tokens[1],tokens[2])).cache()

#Tags\_DF.take(3)

#Spark MLlib for RDDs, Implementing Collaborative filetering using Alternating Least Squares (ALS)

#Dividing Data set for ALS

train\_data, valid\_data, test\_data = Ratings\_DF.randomSplit([6, 2, 2], seed=0L)

validation\_to\_predict = valid\_data.map(lambda x: (x[0], x[1]))

#training on RatingsData

from pyspark.mllib.recommendation import ALS

import math

seed = 5L

iterations = 10

regularization\_parameter = 0.1

ranks = [4,8,12]

errors = [0, 0, 0]

err = 0

tolerance = 0.02

min\_error = float('inf')

best\_rank = -1

best\_iteration = -1

#avg\_error= 0.0

for rank in ranks:

model = ALS.train(train\_data, rank, seed=seed, iterations=iterations,

lambda\_=regularization\_parameter)

predictions = model.predictAll(validation\_to\_predict).map(lambda r: ((r[0], r[1]), r[2]))

rates\_and\_preds = valid\_data.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)

error = math.sqrt(rates\_and\_preds.map(lambda r: (r[1][0] - r[1][1])\*\*2).mean())

errors[err] = error

err += 1

print 'For rank %s the RMSE is %s' % (rank, error)

if error < min\_error:

min\_error = error

best\_rank = rank

print 'In training phase:'

print 'The best model is trained with rank %s' % best\_rank

#Testing Model on Test Data.. here we are removing validation set

train\_data, test\_data = Ratings\_DF.randomSplit([8, 2], seed=5L)

model = ALS.train(train\_data, best\_rank, seed=seed,

iterations=iterations, lambda\_=regularization\_parameter)

test\_to\_predict = test\_data.map(lambda x: (x[0], x[1]))

predictions = model.predictAll(test\_to\_predict).map(lambda r: ((r[0], r[1]), r[2]))

rates\_and\_preds = test\_data.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)

error = math.sqrt(rates\_and\_preds.map(lambda r: (r[1][0] - r[1][1])\*\*2).mean())

print 'In testing phase:'

print 'For testing data the RMSE is %s' % (error)

#Add new user ratings

nuser\_ID = 672

# The format of each line is (userID, movieID, rating)

nuser\_ratings = [

(672,260,4), # Star Wars (1977)

(672,1,3), # Toy Story (1995)

(672,16,3), # Casino (1995)

(672,25,4), # Leaving Las Vegas (1995)

(672,32,4), # Twelve Monkeys (a.k.a. 12 Monkeys) (1995)

(672,335,1), # Flintstones, The (1994)

(672,379,1), # Timecop (1994)

(672,296,3), # Pulp Fiction (1994)

(672,858,5) , # Godfather, The (1972)

(672,50,4) # Usual Suspects, The (1995)

]

new\_user\_ratings= sc.parallelize(nuser\_ratings)

print 'New user ratings: %s' % new\_user\_ratings.take(5)