Import libraries

```
#https://grouplens.org/datasets/movielens/
```

```
import pandas as pd
from pyspark.sql.functions import col, explode
from pyspark import SparkContext
```

Initiate spark session

```
from pyspark.sql import SparkSession
sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Recommendations').getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/01/31 23:35:51 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classe
```

1. Load data

```
movies = spark.read.csv("movies.csv",header=True)
ratings = spark.read.csv("ratings.csv",header=True)
```

```
ratings.show()
```

```
+------+-------+------+---------+
|userId|movieId|rating|timestamp|
+------+-------+------+---------+
|     1|      1|   4.0|964982703|
|     1|      3|   4.0|964981247|
|     1|      6|   4.0|964982224|
|     1|     47|   5.0|964983815|
|     1|     50|   5.0|964982931|
|     1|     70|   3.0|964982400|
|     1|    101|   5.0|964980868|
|     1|    110|   4.0|964982176|
|     1|    151|   5.0|964984041|
|     1|    157|   5.0|964984100|
|     1|    163|   5.0|964983650|
|     1|    216|   5.0|964981208|
|     1|    223|   3.0|964980985|
|     1|    231|   5.0|964981179|
|     1|    235|   4.0|964980908|
|     1|    260|   5.0|964981680|
|     1|    296|   3.0|964982967|
|     1|    316|   3.0|964982310|
|     1|    333|   5.0|964981179|
|     1|    349|   4.0|964982563|
+------+-------+------+---------+
only showing top 20 rows
```

```
ratings.printSchema()
```

```
root
 |-- userId: string (nullable = true)
 |-- movieId: string (nullable = true)
 |-- rating: string (nullable = true)
 |-- timestamp: string (nullable = true)
```

```
ratings = ratings.\
    withColumn('userId', col('userId').cast('integer')).\
    withColumn('movieId', col('movieId').cast('integer')).\
    withColumn('rating', col('rating').cast('float')).\
    drop('timestamp')
ratings.show()
```

```
+------+-------+------+
|userId|movieId|rating|
+------+-------+------+
|     1|      1|   4.0|
|     1|      3|   4.0|
|     1|      6|   4.0|
|     1|     47|   5.0|
|     1|     50|   5.0|
|     1|     70|   3.0|
|     1|    101|   5.0|
|     1|    110|   4.0|
|     1|    151|   5.0|
|     1|    157|   5.0|
|     1|    163|   5.0|
|     1|    216|   5.0|
|     1|    223|   3.0|
|     1|    231|   5.0|
|     1|    235|   4.0|
|     1|    260|   5.0|
|     1|    296|   3.0|
|     1|    316|   3.0|
|     1|    333|   5.0|
|     1|    349|   4.0|
+------+-------+------+
only showing top 20 rows
```

Calculate sparsity

```
# Count the total number of ratings in the dataset
numerator = ratings.select("rating").count()

# Count the number of distinct userIds and distinct movieIds
num_users = ratings.select("userId").distinct().count()
num_movies = ratings.select("movieId").distinct().count()

# Set the denominator equal to the number of users multiplied by the number of movies
denominator = num_users * num_movies

# Divide the numerator by the denominator
sparsity = (1.0 - (numerator *1.0)/denominator)*100
print("The ratings dataframe is ", "%.2f" % sparsity + "% empty.")
```

```
The ratings dataframe is  98.30% empty.
```

Interpret ratings

```
# Group data by userId, count ratings
userId_ratings = ratings.groupBy("userId").count().orderBy('count', ascending=False)
userId_ratings.show()
```

```
+------+-----+
|userId|count|
+------+-----+
|   414| 2698|
|   599| 2478|
|   474| 2108|
|   448| 1864|
|   274| 1346|
|   610| 1302|
|    68| 1260|
|   380| 1218|
|   606| 1115|
|   288| 1055|
|   249| 1046|
|   387| 1027|
|   182|  977|
|   307|  975|
|   603|  943|
|   298|  939|
|   177|  904|
|   318|  879|
|   232|  862|
|   480|  836|
+------+-----+
```

```
only showing top 20 rows
```

```
# Group data by userId, count ratings
movieId_ratings = ratings.groupBy("movieId").count().orderBy('count', ascending=False)
movieId_ratings.show()
```

```
+-------+-----+
|movieId|count|
+-------+-----+
|    356|  329|
|    318|  317|
|    296|  307|
|    593|  279|
|   2571|  278|
|    260|  251|
|    480|  238|
|    110|  237|
|    589|  224|
|    527|  220|
|   2959|  218|
|      1|  215|
|   1196|  211|
|     50|  204|
|   2858|  204|
|     47|  203|
|    780|  202|
|    150|  201|
|   1198|  200|
|   4993|  198|
+-------+-----+
only showing top 20 rows
```

Build Out An ALS Model

```
# Import the required functions
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
```

```
# Create test and train set
(train, test) = ratings.randomSplit([0.8, 0.2], seed = 1234)

# Create ALS model
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", nonnegative = True, implicitPrefs = False, coldStartStrategy=

# Confirm that a model called "als" was created
type(als)
```

```
pyspark.ml.recommendation.ALS
```

Tell Spark how to tune your ALS model

```
# Import the requisite items
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 50, 100, 150]) \
            .addGrid(als.regParam, [.01, .05, .1, .15]) \
            .build()
            #            .addGrid(als.maxIter, [5, 50, 100, 200]) \
```

```
# Define evaluator as RMSE and print length of evaluator
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
print ("Num models to be tested: ", len(param_grid))
```

```
Num models to be tested:  16
```

Build your cross validation pipeline

```
# Build cross validation using CrossValidator
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=5)

# Confirm cv was built
print(cv)
```

```
CrossValidator_a72d78a8012c
```

Best Model and Best Model Parameters

```
#Fit cross validator to the 'train' dataset
model = cv.fit(train)

#Extract best model from the cv model above
best_model = model.bestModel
```

```
24/01/31 23:35:56 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
24/01/31 23:35:56 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.VectorBLAS
24/01/31 23:36:05 WARN GarbageCollectionMetrics: To enable non-built-in garbage collector(s) List(G1 Concurrent GC), users s
```

```
# Print best_model
print(type(best_model))

# Complete the code below to extract the ALS model parameters
print("**Best Model**")

# # Print "Rank"
print("  Rank:", best_model._java_obj.parent().getRank())

# Print "MaxIter"
print("  MaxIter:", best_model._java_obj.parent().getMaxIter())

# Print "RegParam"
print("  RegParam:", best_model._java_obj.parent().getRegParam())
```

```
<class 'pyspark.ml.recommendation.ALSModel'>
**Best Model**
  Rank: 50
  MaxIter: 10
  RegParam: 0.15
```

```
# View the predictions
test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print(RMSE)
```

```
0.8677116560030962
```

```
test_predictions.show()
```

```
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|   148|    356|   4.0| 3.4877493|
|   148|   4896|   4.0| 3.4659078|
|   148|   4993|   3.0|  3.506965|
|   148|   7153|   3.0| 3.4264944|
|   148|   8368|   4.0| 3.5371032|
|   148|  40629|   5.0| 3.2190442|
|   148|  50872|   3.0| 3.7063406|
|   148|  60069|   4.5| 3.6646976|
|   148|  69757|   3.5| 3.4481666|
|   148|  72998|   4.0| 3.2189465|
|   148|  81847|   4.5|  3.517452|
|   148|  98491|   5.0|  3.741063|
|   148| 115617|   3.5| 3.5368829|
|   148| 122886|   3.5| 3.4820495|
|   463|    296|   4.0|  4.184284|
|   463|    527|   4.0|  3.820791|
|   463|   2019|   4.0|  3.962076|
```

```
|   471|    527|   4.5| 3.8060656|
|   471|   6016|   4.0| 3.9264812|
|   471|   6333|   2.5| 3.2160175|
+------+-------+------+----------+
only showing top 20 rows
```

Make Recommendations

```
# Generate n Recommendations for all users
nrecommendations = best_model.recommendForAllUsers(10)
nrecommendations.limit(10).show()
```

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|     1|[{3379, 5.7274647...|
|     2|[{131724, 4.79768...|
|     3|[{6835, 4.8469415...|
|     4|[{3851, 4.849598}...|
|     5|[{3379, 4.5703735...|
|     6|[{33649, 4.744535...|
|     7|[{3379, 4.5015903...|
|     8|[{3379, 4.7354617...|
|     9|[{3379, 4.7853327...|
|    10|[{71579, 4.53796}...|
+------+--------------------+
```

```
nrecommendations = nrecommendations\
    .withColumn("rec_exp", explode("recommendations"))\
    .select('userId', col("rec_exp.movieId"), col("rec_exp.rating"))

nrecommendations.limit(10).show()
```

```
+------+-------+---------+
|userId|movieId|   rating|
+------+-------+---------+
|     1|   3379|5.7274647|
|     1|   5490|  5.62538|
|     1|  33649| 5.556481|
|     1|   5915|  5.43931|
|     1| 171495|5.4147716|
|     1|   5416| 5.395296|
|     1|   5328| 5.395296|
|     1|   3951| 5.395296|
|     1|  78836| 5.368873|
|     1|   6460| 5.349636|
+------+-------+---------+
```

Do the recommendations make sense?

Lets merge movie name and genres to teh recommendation matrix for interpretability.

```
nrecommendations.join(movies, on='movieId').filter('userId = 100').show()
```

```
+-------+------+---------+--------------------+--------------------+
|movieId|userId|   rating|               title|              genres|
+-------+------+---------+--------------------+--------------------+
|  67618|   100| 5.085453|Strictly Sexual (...|Comedy|Drama|Romance|
|   3379|   100|5.0430155| On the Beach (1959)|               Drama|
|  33649|   100| 4.982005|  Saving Face (2004)|Comedy|Drama|Romance|
|  42730|   100| 4.981763|   Glory Road (2006)|               Drama|
|  74282|   100|4.9055843|Anne of Green Gab...|Children|Drama|Ro...|
| 171495|   100| 4.870512|              Cosmos|  (no genres listed)|
| 184245|   100|4.8490896|De platte jungle ...|         Documentary|
| 179135|   100|4.8490896|Blue Planet II (2...|         Documentary|
| 138966|   100|4.8490896|Nasu: Summer in A...|           Animation|
| 117531|   100|4.8490896|    Watermark (2014)|         Documentary|
+-------+------+---------+--------------------+--------------------+
```

```
ratings.join(movies, on='movieId').filter('userId = 100').sort('rating', ascending=False).limit(10).show()
```

```
+-------+------+------+--------------------+--------------------+
|movieId|userId|rating|               title|              genres|
+-------+------+------+--------------------+--------------------+
|   1101|   100|   5.0|      Top Gun (1986)|      Action|Romance|
|   1958|   100|   5.0|Terms of Endearme...|        Comedy|Drama|
|   2423|   100|   5.0|Christmas Vacatio...|              Comedy|
|   4041|   100|   5.0|Officer and a Gen...|       Drama|Romance|
|   5620|   100|   5.0|Sweet Home Alabam...|      Comedy|Romance|
|    368|   100|   4.5|     Maverick (1994)|Adventure|Comedy|...|
|    934|   100|   4.5|Father of the Bri...|              Comedy|
|    539|   100|   4.5|Sleepless in Seat...|Comedy|Drama|Romance|
|     16|   100|   4.5|       Casino (1995)|         Crime|Drama|
|    553|   100|   4.5|    Tombstone (1993)|Action|Drama|Western|
+-------+------+------+--------------------+--------------------+
```