# AWS AI/ML Workshop on Recommendation Engine using Apache Spark ML with Spark’s DataFrames API

### Starting with the Notebook

1. Click on **Create new note.**
2. A dialog box will appear where you will provide a name for the Notebook, name it anything you feel like and choose the Default Interpreter to be **spark.**
3. Click the **Create Note** button.

#### The imports,

We import **SparkSession** and **DataFrame** for the obvious reason, we import **SQLContext** so that we can instantiate one to use for registering tables off a DataFrame and also, if required to use SQL queries. You can see their respective Python documentation at [this link](https://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html) . We import **functions** to enable us to apply functions to full columns, you can find the details of functions in the same link as previously provided. We use **ALS** to train on the training dataset and hence the import of ALS. The RegressionEvaluator is imported to enable us to measure the quality of our model, we use the RMSE (Root Mean Squared Error) to evaluate our model, a very common approach for model evaluation. You can find the details of ALS and RegressionEvaluator [here](https://spark.apache.org/docs/2.1.0/api/python/pyspark.ml.html).

%pyspark

from pyspark.sql import SparkSession,DataFrame,Column,SQLContext

from pyspark.sql import functions as Fun

from pyspark.ml.recommendation import ALS

from pyspark.ml.evaluation import RegressionEvaluator

import pyspark

import os,sys

#### Choose your source and destination buckets,

We have provided a bucket for Read-Only resources, your code will pull all necessary data for you to get started from this bucket, you will be required to create a bucket, if not already created, for storing your personal ratings, which you should have run per the workshop conductor’s instructions, if you haven’t already and you were not instructed to, you can ask the same from the person conducting the workshop. You should be able to write to your bucket and also read from it.

%pyspark

ROsrcBkt = "s3://**myBucket**/" # For all the general input data for this workshop

MydestBkt = "s3://**myBucket**/[**Prefix**]" # This is where you have stored your personal ratings

#### Read the ratings you have from the input file, ratings.csv

These lines enable you to read the ratings CSV file from the S3 location previously mentioned (see above). The expected result of these lines is the creation of a DataFrame from the data in the CSV file.

%pyspark

ratingsDF = spark.read.option("header","true").csv(ROsrcBkt+"ratings.csv")

#### Split data into training, validation and test data sets,

Here, we split data the data into training, validation and test datasets. The training data will be used for training while the validation data will be used for validation of the model for unseen inputs. You may decide to skip the creation of the validation dataset as the future calls to train the model that we shall make provide a k-folds cross-validator to come up with and optimal model. If you choose to only proceed with a training and tests set, you will have to make minor adjustments to the code that follows.

%pyspark

seed = 42

(trngDF,valDF,testDF) = ratingsDF.randomSplit([0.6,0.2,0.2],seed)

trngDF.persist()

valDF.persist()

testDF.persist()

#### Read your personal ratings and add them to the training data set for developing the model,

Here we read our personal ratings and create a DataFrame out of it, we then union the training DataFrame with the personal ratings DataFrame, finally we fix the DataFrame by setting the correct data types for their columns.

%pyspark

myRatingsDF = spark.read.option("header","false").csv(MydestBkt+"personalRatings.txt")

myRatingsDF.persist()

nTrngDF = trngDF.union(myRatingsDF.drop('timestamp'))

nnTrngDF = nTrngDF.select(nTrngDF.userId.cast("int"),nTrngDF.movieId.cast("int"),nTrngDF.rating.cast("float"))

nnTrngDF.persist()

trngDF.unpersist()

nValDF = valDF.select(valDF.userId.cast("int"),valDF.movieId.cast("int"),valDF.rating.cast("float"))

nValDF.persist()

nTestDF = testDF.select(testDF.userId.cast("int"),testDF.movieId.cast("int"),testDF.rating.cast("float"))

nTestDF.persist()

valDF.unpersist()

testDF.unpersist()

nnTrngDF.printSchema()

nValDF.printSchema()

nTestDF.printSchema()

#### Set up your parameters,

Note: Choose any values you prefer, for the purposes of the workshop, remember that higher rank and more iterations tend to result in longer learning cycles, means that it requires more time and more resources. Kindly be mindful of this during the workshop.

%pyspark

irank = [21,22,23]

iterations = 20

ireg = 0.2

ilearn = 1.0

model = None

oldrmse = None

#### Identify a suitable ML model to use for future predictions,

You can try different values of the hyper-parameters (see **Setup your parameters** above) to get the least possible Root Mean Squared Error (RMSE) value for your model/s

%pyspark

for r in irank:

newmodel = None

alsresult = ALS(rank=r,maxIter=iterations,regParam=ireg,alpha=ilearn,userCol="userId",itemCol="movieId",ratingCol="rating")

newmodel = alsresult.fit(nnTrngDF)

predictions = newmodel.transform(nValDF)

evaluator = RegressionEvaluator(metricName="rmse",labelCol="rating",predictionCol="prediction")

rmse = evaluator.evaluate(predictions.filter(predictions.prediction != float('nan')))

print("Found model with Root Mean Squared Error of {0:3.4f}.".format(rmse))

if not model:

model = newmodel

oldrmse = rmse

else:

if oldrmse > rmse:

model = newmodel

oldrmse = rmse

nnTrngDF.unpersist()

nValDF.unpersist()

print("Identified best model whose Root Mean Squared Error on Validation data is {0:3.4f}.".format(oldrmse))

#### Check your model on the test data,

Again, if you are not happy with the RMSE you get on the test data, try different hyper parameters as suggested above.

%pyspark

predictions = model.transform(nTestDF)

evaluator = RegressionEvaluator(metricName="rmse",labelCol="rating",predictionCol="prediction")

oldrmse = evaluator.evaluate(predictions.filter(predictions.prediction != float('nan')))

print("The best model we have chosen gives us a RMSE of {0:3.4f} on our test dataset.".format(oldrmse))

#### Compare the model with an average rating model,

Your model should give a lower RMSE compared to the average rating model, you may want to revise your hyper parameters if the average model beats your model.

%pyspark

trng\_avg\_rating = nnTrngDF.agg(Fun.avg(Fun.col("rating"))).alias("avg").collect()

nAvgTestDF = nTestDF.withColumn("prediction",Fun.lit(trng\_avg\_rating[0][0]))

nTestDF.unpersist()

print("Checking the RMSE for the average rating: {0:3.4f}".format(evaluator.evaluate(nAvgTestDF.filter(nAvgTestDF.prediction != float('nan')))))

#### Get your recommendations,

Here we get recommendations for movies that we have not seen and or rated.

%pyspark

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*Top 20 recommended movies for you:\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

allMoviesDF = spark.read.option("header","true").csv(ROsrcBkt+"movies.csv")

nallMoviesDF = (allMoviesDF.drop('genres').drop('title'))

nmyRatingsDF = myRatingsDF.drop('\_c0').drop('\_c2').drop('\_c3').withColumn("userId",Fun.lit(0))

nnmyRatingsDF = nmyRatingsDF.select(nmyRatingsDF.\_c1.cast("int"))

myRatingsDF.unpersist()

nnallMoviesDF = nallMoviesDF.select(nallMoviesDF.movieId.cast("int"))

unratedMoviesDF = nnallMoviesDF.subtract(nnmyRatingsDF)

nunratedMoviesDF = unratedMoviesDF.withColumn("userId",Fun.lit(0))

nnunratedMoviesDF = nunratedMoviesDF.select('userId','movieId')

predictedRatingsDF = model.transform(nnunratedMoviesDF)

npredictedRatingsDF = (predictedRatingsDF.filter(predictedRatingsDF['prediction'] != float('nan'))).orderBy("prediction",ascending=False)

for data in npredictedRatingsDF.take(20):

print("{}".format((allMoviesDF.select("title").filter(data.movieId == allMoviesDF.movieId).collect())[0]['title'].encode("utf-8")))

\*\*\*Note: You should see your recommendations change as you change your personal ratings.