## Part A:

1. [Marks: 10] Count the odd and even numbers using file 'integer.txt' download it from the Quercus. Show your code and output.

```
▶ ✓ Just now (<1s)
                                                                                                                             Python []
  import numpy as np
 import pandas as pd
 from pyspark.sql import SparkSession
 from pyspark.sql.functions import col
 from pyspark.sql.types import IntegerType,FloatType
 from pyspark.sql import types
 from pyspark.sql import functions as F
 from pyspark import SparkContext
 from pyspark.ml.recommendation import ALS
 from pyspark.ml.evaluation import RegressionEvaluator
 from pyspark.ml.tuning import ParamGridBuilder, CrossValidator, TrainValidationSplit
       ✓ 12:11 AM (2s)
 spark = SparkSession.builder.appName('counter').getOrCreate()
 data= spark.read.text("dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/integer.txt")
 data=data.withColumn("value", col('value').cast(IntegerType()))
 count_even=data.filter(data.value % 2==0).count()
 count_odd = data.filter(data.value % 2!=0).count()
 print("The count of odd number is: ",count_odd)
 print("THe count of even number is : ", count even)
▶ ■ data: pyspark.sql.dataframe.DataFrame = [value: integer]
The count of odd number is: 496
THe count of even number is : 514
```

2. [Marks: 10] Calculate the salary sum per department using file 'salary.txt' download it from the Quercus. Show department name and salary sum. Show your code and output.

```
Python []
 \verb|#Question2| dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/salary.txt|
  spark = SparkSession.builder.appName('sum_of_salary').getOrCreate()
 data = spark.read.text("dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/salary.txt")
 data=data.withColumn("department", F.split(F.col('value'),' ')[0])
 data=data.withColumn("salary", F.split(F.col('value'),' ')[1].cast(FloatType()))
 data_department_group= data.groupby("department").sum()
 data_department_group.show()
▶ (2) Spark Jobs
▶ 🗐 data: pyspark.sql.dataframe.DataFrame = [value: string, department: string ... 1 more field]
tala_department_group: pyspark.sql.dataframe.DataFrame = [department: string, sum(salary): double]
|department|sum(salary)|
    Sales| 3488491.0|
| Developer| 3221394.0|
| Research| 3328284.0|
| Marketing | 3158450.0|
      QA| 3360624.0|
```

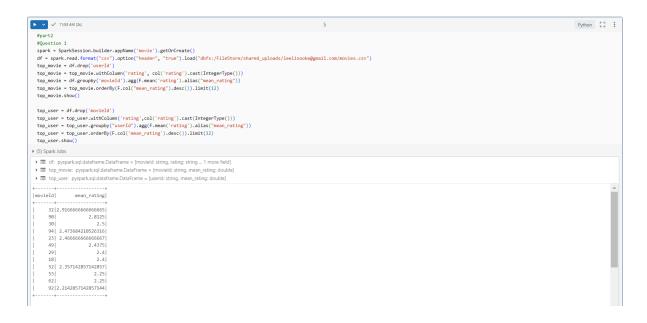
3. [Marks: 10] Implement MapReduce using Pyspark on file 'shakespeare.txt' download it from the Quercus. Show how many times these particular words appear in the document: Shakespeare, When, Lord, Library, GUTENBERG, WILLIAM, COLLEGE and WORLD. (Count exact words only)

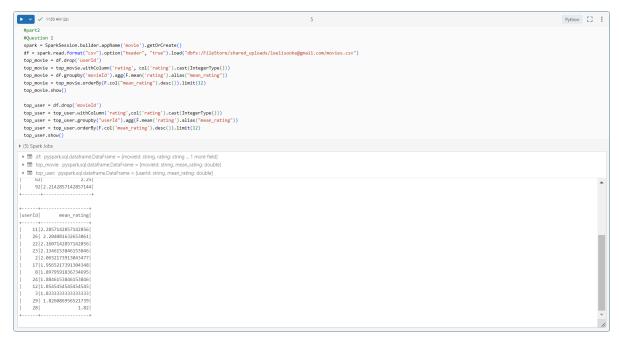
4. [Marks: 10] Calculate top 15 and bottom 15 words using file 'shakespeare.txt' download it from the Quercus. Show 15 words with most count and 15 words with least count. You can limit by 15 in ascending and descending order of count. Show your code and output

```
Python []
  #Question 3 dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/shakespeare_1.txt
  spark = SparkSession.builder.appName('word_count').getOrCreate()
  data = spark.sparkContext.textFile("dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/shakespeare_1.txt")
  counts= data.flatMap(lambda line : line.split(" "))\
   .map(lambda word: (word,1))\
        .reduceBvKev(lambda a, b: a+b)
 words= ["Shakespeare","When","Lord", "Library", "GUTENBERG","WILLIAM","COLLEGE","WORLD"]
  words_list= counts.filter(lambda x: x[0] in words)
 print("Word counts: ",words_list.collect())
 print('\n')
  top_words= counts.takeOrdered(15,key=lambda x : -x[1])
 bot words= counts.takeOrdered(15, key=lambda x: x[1])
 print("Top 15 words: ",top_words)
 print("Bottom 15 words: ", bot words)
Word counts: [('Shakespeare', 22), ('GUTENBERG', 99), ('WILLIAM', 115), ('WORLD', 98), ('COLLEGE', 98), ('When', 393), ('Lord', 341), ('Library', 2)]
Top 15 words: [('', 231583), ('the', 11397), ('and', 8777), ('I', 8556), ('of', 7873), ('to', 7421), ('a', 5672), ('my', 4913), ('in', 4600), ('you', 4060), ('And',
3547), ('that', 3522), ('is', 3481), ('his', 3226), ('with', 3175)] bottom 15 words: [('anyone', 1), ('restrictions', 1), ('whatsoever.', 1), ('re-use', 1), ('online', 1), ('www.gutenberg.org', 1), ('COPYRIGHTED', 1), ('eBook,', 1),
('Details', 1), ('guidelines', 1), ('file.', 1), ('Author:', 1), ('Posting', 1), ('1,', 1), ('2011', 1)]
```

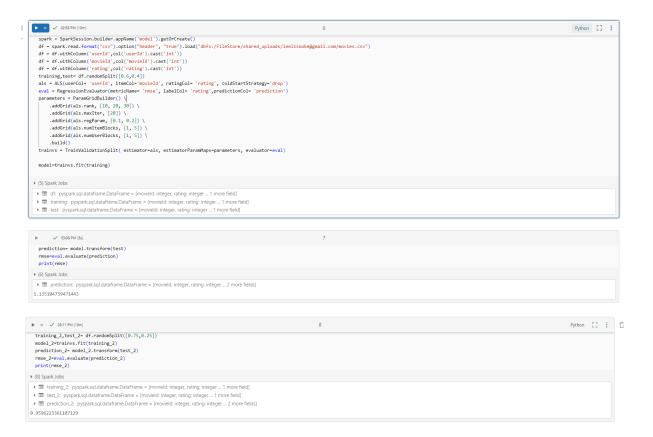
## Part 2:

1. [Marks: 10] Describe your data. Calculate top 12 movies with highest ratings and top 12 users who provided highest ratings. Show your code and output





2. [Marks: 10] Split dataset into train and test. Try 2 different combinations for e.g. (60/40, 70/30, 75/25 and 80/20). (Train your model and use collaborative filtering approach on 70 percent of your data and test with the other 30 percent and so on). Show your code and output



3. [Marks: 10] Explain MSE, RMSE and MAE. Compare and evaluate both of your models with evaluation metrics (RMSE or MAE), show your code and print your results. Describe which one works better and why?

MSE is the average of the squared differences between the actual values and the predicted values. RMSE is the square root of the MSE. MAE is the average of the absolute differences between the actual and predicted values.

```
Python [] : []

rmse_eva = RegressionEvaluator(metricllame="rese", labelCol="rating", predictionCol="prediction")
rmse_l = rmse_eva.evaluate(prediction)
rmse_2 = rmse_eva.evaluate(prediction)
mae_2 = rmse_eva.evaluate(prediction)
mae_2 = rmse_eva.evaluate(prediction)
mae_2 = rmse_eva.evaluate(prediction)

print(f*Model 1 - BMSE: (rmse_1), MAE: (mae_1)")
print(f*Model 2 - BMSE: (rmse_2), MAE: (mae_2)")

* (8) Spark Jobs

Model 1 - BMSE: 1.135184739471443, MAE: 0.7754954818894367
Model 2 - BMSE: 0.9596223361107129, MAE: 0.6580533112224182
```

MAE's results are relatively better than RMSE. I think this is because MAE does not penalize outliers or occasional large errors as much as RMSE does. RMSE penalizes those outliers by scaling the error, which is why the RMSE result is relatively higher than that of MAE in this case.

4. [Marks: 20] Now tune the parameters of your algorithm to get the best set of parameters. Explain different parameters of the algorithm which you have used for tuning your algorithm. Evaluate all your models again. Show your code with best values and output

rank: rank controls the number of features used to represent users and items.

maxIter: The maximum number of iterations the ALS algorithm will run regParam: The regularization parameter to control overfitting. A higher value of regParam can reduce overfitting by penalizing large model parameters, but it may also underfit the data if set too high.

```
#Question 4

parameters_2 = ParamGridBuilder() \

addGrid(als.rank, [10, 20, 30]) \
addGrid(als.maxIter, [10, 20, 30]) \
addGrid(als.maxIter, [10, 20, 30]) \
addGrid(als.numItemBlocks, [1, 5]) \
addGrid(als.numItemBlocks, [1, 5]) \
build()

trainvs_2 = TrainValidationSplit( estimator=als, estimatorParamMaps=parameters_2, evaluator=eval)

model_3=trainvs_2.fit(training_2)

(5) Spark Jobs
```

```
▶ ✓ 1 minute ago (15s)
  best model = model_3.bestModel
  prediction_3= best_model.transform(test_2)
  rmse_3=eval.evaluate(prediction_3)
  print("rmse of best model: ",rmse 3)
  print("Best rank:", best_model.rank)
 print("Best maxIter:", best_model._java_obj.parent().getMaxIter())
print("Best regParam:", best_model._java_obj.parent().getRegParam())
  rmse eva = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
  rmse_1 = rmse_eva.evaluate(prediction)
  rmse_2 = rmse_eva.evaluate(prediction_2)
  rmse 3 = rmse eva.evaluate(prediction 3)
  mae eva = RegressionEvaluator(metricName="mae", labelCol="rating", predictionCol="prediction")
  mae_1 = mae_eva.evaluate(prediction)
  mae_2 = mae_eva.evaluate(prediction_2)
  mae 3 = mae eva.evaluate(prediction 3)
  print(f"Model 1 - RMSE: {rmse_1}, MAE: {mae_1}")
  print(f"Model 2 - RMSE: {rmse_2}, MAE: {mae_2}")
  print(f"Model 3 - RMSE: {rmse_3}, MAE: {mae_3}")
▶ (4) Spark Jobs
 ▶ ■ prediction_3: pyspark.sql.dataframe.DataFrame = [movield: integer, rating: integer ... 2 more fields]
rmse of best model: 0.948573238537854
Best maxIter: 10
Best regParam: 0.1
Model 1 - RMSE: 1.135104739471443, MAE: 0.7754954818894367
Model 2 - RMSE: 0.9596223361107129, MAE: 0.6580533112224182
Model 3 - RMSE: 0.948573238537854, MAE: 0.6538727079957155
```

5. [Marks: 10]: Calculate top 12 movies recommendations for user id 10 and user id 12. Show your code and output.