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) 1 Python

```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) 2

```python
#Question1
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)
```

(4) Spark Jobs

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.

12:21 AM (1s) 3 Python

```python
#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]

data_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)

▸ ⌄ ✓ 01:03 AM (2s) 4 Python

```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")
#mapreduce
counts= data.flatMap(lambda line : line.split(" "))\
    .map(lambda word: (word,1))\
        .reduceByKey(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())

#Question 4:
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)
```

▸ (3) Spark Jobs

```
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)]
```

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

▸ ⌄ ✓ 01:03 AM (2s) 4 Python

```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")
#mapreduce
counts= data.flatMap(lambda line : line.split(" "))\
    .map(lambda word: (word,1))\
        .reduceByKey(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())

#Question 4:
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)
```

▸ (3) Spark Jobs

```
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

11:53 AM (2s) 5 Python

```python
#part2
#Question 1
spark = SparkSession.builder.appName('movie').getOrCreate()
df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/movies.csv")
top_movie = df.drop('userId')
top_movie = top_movie.withColumn('rating', col('rating').cast(IntegerType()))
top_movie = df.groupby('movieId').agg(F.mean('rating').alias("mean_rating"))
top_movie = top_movie.orderBy(F.col("mean_rating").desc()).limit(12)
top_movie.show()

top_user = df.drop('movieId')
top_user = top_user.withColumn('rating',col('rating').cast(IntegerType()))
top_user = top_user.groupby("userId").agg(F.mean('rating').alias("mean_rating"))
top_user = top_user.orderBy(F.col('mean_rating').desc()).limit(12)
top_user.show()
```

(5) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [movieId: string, rating: string ... 1 more field]
top_movie: pyspark.sql.dataframe.DataFrame = [movieId: string, mean_rating: double]
top_user: pyspark.sql.dataframe.DataFrame = [userId: string, mean_rating: double]

```
+-------+------------------+
|movieId|       mean_rating|
+-------+------------------+
|     32|2.9166666666666665|
|     90|            2.8125|
|     30|               2.5|
|     94| 2.473684210526316|
|     23| 2.466666666666667|
|     49|            2.4375|
|     29|               2.4|
|     18|               2.4|
|     52| 2.357142857142857|
|     53|              2.25|
|     62|              2.25|
|     92|2.2142857142857144|
+-------+------------------+
```

11:53 AM (2s) 5 Python

```python
#part2
#Question 1
spark = SparkSession.builder.appName('movie').getOrCreate()
df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/movies.csv")
top_movie = df.drop('userId')
top_movie = top_movie.withColumn('rating', col('rating').cast(IntegerType()))
top_movie = df.groupby('movieId').agg(F.mean('rating').alias("mean_rating"))
top_movie = top_movie.orderBy(F.col("mean_rating").desc()).limit(12)
top_movie.show()

top_user = df.drop('movieId')
top_user = top_user.withColumn('rating',col('rating').cast(IntegerType()))
top_user = top_user.groupby("userId").agg(F.mean('rating').alias("mean_rating"))
top_user = top_user.orderBy(F.col('mean_rating').desc()).limit(12)
top_user.show()
```

(5) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [movieId: string, rating: string ... 1 more field]
top_movie: pyspark.sql.dataframe.DataFrame = [movieId: string, mean_rating: double]
top_user: pyspark.sql.dataframe.DataFrame = [userId: string, mean_rating: double]

```
|     62|              2.25|
|     92|2.2142857142857144|
+-------+------------------+
```


```
+------+------------------+
|userId|       mean_rating|
+------+------------------+
|    11|2.2857142857142856|
|    26| 2.204081632653061|
|    22|2.1607142857142856|
|    23|2.1346153846153846|
|     2|2.0652173913043477|
|    17|1.9565217391304348|
|     8|1.8979591836734695|
|    24|1.8846153846153846|
|    12|1.8545454545454545|
|     3|1.8333333333333333|
|    29| 1.826086956521739|
|    28|              1.82|
+------+------------------+
```

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

```
spark = SparkSession.builder.appName('model').getOrCreate()
df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_uploads/leelisooke@gmail.com/movies.csv")
df = df.withColumn('userId',col('userId').cast('int'))
df = df.withColumn('movieId',col('movieId').cast('int'))
df = df.withColumn('rating',col('rating').cast('int'))
training,test= df.randomSplit([0.6,0.4])
als = ALS(userCol= 'userId', itemCol='movieId', ratingCol= 'rating', coldStartStrategy='drop')
eval = RegressionEvaluator(metricName= 'rmse', labelCol= 'rating',predictionCol= 'prediction')
parameters = ParamGridBuilder() \
    .addGrid(als.rank, [10, 20, 30]) \
    .addGrid(als.maxIter, [20]) \
    .addGrid(als.regParam, [0.1, 0.2]) \
    .addGrid(als.numItemBlocks, [1, 5]) \
    .addGrid(als.numUserBlocks, [1, 5]) \
    .build()
trainvs = TrainValidationSplit( estimator=als, estimatorParamMaps=parameters, evaluator=eval)

model=trainvs.fit(training)
```

```
(5) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
training: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
test: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
```

```
prediction= model.transform(test)
rmse=eval.evaluate(prediction)
print(rmse)
```

```
(6) Spark Jobs
prediction: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
1.135104739471443
```

```
training_2,test_2= df.randomSplit([0.75,0.25])
model_2=trainvs.fit(training_2)
prediction_2= model_2.transform(test_2)
rmse_2=eval.evaluate(prediction_2)
print(rmse_2)
```

```
(8) Spark Jobs
training_2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
test_2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
prediction_2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
0.9596223361107129
```

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.

```
rmse_eva = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse_1 = rmse_eva.evaluate(prediction)
rmse_2 = rmse_eva.evaluate(prediction_2)

mae_eva = RegressionEvaluator(metricName="mae", labelCol="rating", predictionCol="prediction")
mae_1 = mae_eva.evaluate(prediction)
mae_2 = mae_eva.evaluate(prediction_2)

print(f"Model 1 - RMSE: {rmse_1}, MAE: {mae_1}")
print(f"Model 2 - RMSE: {rmse_2}, MAE: {mae_2}")
```

```
(8) Spark Jobs
Model 1 - RMSE: 1.135104739471443, MAE: 0.7754954818894367
Model 2 - RMSE: 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.

03:45 PM (54m) 10 Python

```python
#Question 4
parameters_2 = ParamGridBuilder() \
    .addGrid(als.rank, [10, 20, 30]) \
    .addGrid(als.maxIter, [10,20,30]) \
    .addGrid(als.regParam, [0.1, 0.2,0.3]) \
    .addGrid(als.numItemBlocks, [1, 5]) \
    .addGrid(als.numUserBlocks, [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) 11 Python

```python
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 = [movieId: integer, rating: integer ... 2 more fields]

```
rmse of best model:  0.948573238537854
Best rank: 10
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.

Just now (3s) 12 Python

```python
#Question 5
user_ids = spark.createDataFrame([(10,), (12,)], ["userId"])
user_rec= best_model.recommendForUserSubset(user_ids,12)
print(user_rec.show(truncate=False))
```

(5) Spark Jobs

user_ids: pyspark.sql.dataframe.DataFrame = [userId: long]
user_rec: pyspark.sql.dataframe.DataFrame = [userId: integer, recommendations: array]

```
+------+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
|userId|recommendations
|
+------+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
|12    |[{17, 4.4534}, {48, 4.315681}, {27, 4.1103806}, {32, 3.978695}, {90, 3.8604753}, {35, 3.7725995}, {46, 3.5740008}, {94, 3.5378063}, {18, 3.534183}, {50, 3.502
1763}, {16, 3.4162762}, {55, 3.3012254}]|
|10    |[{92, 3.427152}, {2, 3.161942}, {25, 2.8951662}, {89, 2.8869176}, {40, 2.862807}, {49, 2.7971985}, {12, 2.662564}, {42, 2.5345876}, {62, 2.5064726}, {0, 2.281
7194}, {95, 2.1507835}, {9, 2.0935311}] |
+------+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+

None
```