All result code and output can be seen either in attached jupyter notebook or at the end of this PDF.

Part A:

1. Count of odd and even numbers.

```
#reading in text file
sourceRDD_A1 = spark.sparkContext.textFile("/FileStore/tables/integer.txt", 4)
```

Command took 0.06 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

Cmd 7

```
#counting total number of records in RDD
sourceRDD_A1.count()
```

▸ (1) Spark Jobs

```
Out[237]: 1010
```

Command took 0.84 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

Cmd 8

```
#Transformation
#converting x to floats
float_RDD = sourceRDD_A1.map(lambda x: float(x))
```

Command took 0.04 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

Cmd 9

```
#Transformation
#operations to filter for even and odd numbers
even_RDD = float_RDD.filter(lambda x: x%2 == 0)
odd_RDD = float_RDD.filter(lambda x: x%2 != 0)
```

Command took 0.04 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

Cmd 10

```
#Action
#Number of even numbers
even_RDD.count()
```

▸ (1) Spark Jobs

```
Out[240]: 514
```

Command took 0.33 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

Cmd 11

```
#Action
#Number of odd numbers
odd_RDD.count()
```

▸ (1) Spark Jobs

```
Out[241]: 496
```

2. Salary sum for each department.

```
#reading in text file
sourceRDD_A2 = spark.sparkContext.textFile("/FileStore/tables/salary.txt", 4)
#viewing source rdd
sourceRDD_A2.collect()
```

▸ (1) Spark Jobs

```
Out[6]: ['Sales 9136',
 'Research 13391',
 'Developer 22220',
 'QA 31888',
 'Marketing 22215',
```

Command took 0.89 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 2:25:43 PM on Cluster_Assing2_1

Cmd 10

```
#creating a list of arrays for each row
arrayRDD_A2 = sourceRDD_A2.map(lambda x: x.split(" "))
arrayRDD_A2.collect()
```

▸ (1) Spark Jobs

```
Out[7]: [['Sales', '9136'],
 ['Research', '13391'],
 ['Developer', '22220'],
 ['QA', '31888'],
 ['Marketing', '22215'],
```

Command took 0.66 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 2:25:59 PM on Cluster_Assing2_1

Cmd 11

```
#converting the 2nd element (position 1) into float
kvRDD_A2 = arrayRDD_A2.map(lambda x: (x[0], float(x[1])))
kvRDD_A2.collect()
```

▸ (1) Spark Jobs

```
Out[8]: [('Sales', 9136.0),
 ('Research', 13391.0),
 ('Developer', 22220.0),
 ('QA', 31888.0),
 ('Marketing', 22215.0),
```

Command took 0.72 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 2:26:49 PM on Cluster_Assing2_1

Cmd 12

```
#calculating the salary sum for each department
sumRDD_A2 = kvRDD_A2.reduceByKey(lambda x,y: x+y)
#Salary sum for each department
sumRDD_A2.collect()
```

▸ (1) Spark Jobs

```
Out[9]: [('Developer', 3221394.0),
 ('Sales', 3488491.0),
 ('Research', 3328284.0),
 ('QA', 3360624.0),
 ('Marketing', 3158450.0)]
```

Command took 1.76 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 2:27:06 PM on Cluster_Assing2_1

3. MapReduce using pyspark.

```
#MapReduce -> Mapping step
mapper_RDD_A3 = input_RDD_A3.map(lambda word: (word, 1))
mapper_RDD_A3.collect()
```

▸ (1) Spark Jobs

```
Out[14]: [('This', 1),
 ('eBook', 1),
 ('is', 1),
 ('for', 1),
 ('the', 1),
```

Command took 2.75 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 2:31:23 PM on Cluster_Assing2_1

Cmd 19

```
#MapReduce -> Reducer step
reducer_RDD_A3 = mapper_RDD_A3.reduceByKey(lambda x, y: x + y)
reducer_RDD_A3.collect()
```

▸ (1) Spark Jobs

```
Out[15]: [('of', 7957),
 ('no', 1270),
 ('whatsoever', 8),
 ('may', 642),
 ('give', 473),
```

Command took 3.06 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 2:32:02 PM on Cluster_Assing2_1

```
#convert to dataframe and cache
wordcounts = spark.createDataFrame(reducer_RDD_A3).cache()
#changing column headers of wordcount dataframe
wordcounts = wordcounts.select(col("_1").alias("word"), col("_2").alias("count"))
wordcounts.show()
```

▸ (2) Spark Jobs

▸ wordcounts: pyspark.sql.dataframe.DataFrame = [word: string, count: long]

```
+-----------+-----+
|       word|count|
+-----------+-----+
|         of| 7957|
|         no| 1270|
```

Command took 2.92 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 2:32:20 PM on Cluster_Assing2_1

Cmd 21

```
#filtering the dataframe for the EXACT words in word_list
word_list = ['Shakespeare', 'When', 'Lord', 'Library', 'GUTENBERG', 'WILLIAM', 'COLLEGE', 'WORLD']
filtered_words = wordcounts.filter(func.col('word').rlike('(^|\s)(' + '|'.join(word_list) + ')(\s|$)'))
#result
filtered_words.show()
```

▸ (2) Spark Jobs

▸ filtered_words: pyspark.sql.dataframe.DataFrame = [word: string, count: long]

```
+-----------+-----+
|       word|count|
+-----------+-----+
|Shakespeare|   22|
|    WILLIAM|  127|
|      WORLD|   98|
|       When|  405|
|  GUTENBERG|   99|
|    COLLEGE|   98|
|       Lord|  400|
|    Library|    5|
+-----------+-----+
```

4. Word Count.

Top 15

```
#Top 15 words with the most count
wordcounts.orderBy('count', ascending=False).show(15)
```

▸ (1) Spark Jobs

```
+----+-----+
|word|count|
+----+-----+
| the|11402|
|   I| 8936|
| and| 8916|
|  of| 7957|
|  to| 7508|
|   a| 5755|
| you| 5306|
|  my| 4919|
|  in| 4722|
|that| 3754|
| And| 3735|
|  is| 3666|
| not| 3588|
|  me| 3445|
| his| 3278|
+----+-----+
only showing top 15 rows
```

Bottom 15

```
#Bottom 15 words with the least count
wordcounts.orderBy('count', ascending=True).show(15)
```

▸ (1) Spark Jobs

```
+----------+-----+
|      word|count|
+----------+-----+
|provisions|    1|
|INCIDENTAL|    1|
|processing|    1|
|       NOR|    1|
|    EITHER|    1|
|   Release|    1|
|     tilde|    1|
|    CDROMS|    1|
|    reader|    1|
|     Title|    1|
|    EBCDIC|    1|
|     EBOOK|    1|
|equivalent|    1|
|      AWAY|    1|
|   program|    1|
+----------+-----+
only showing top 15 rows
```

Part B:

1. Exploratory Data Analysis.
   a. Null Values

```
#checking for any null values
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

▸ (2) Spark Jobs

```
+-------+------+------+
|movieId|rating|userId|
+-------+------+------+
|      0|     0|     0|
+-------+------+------+
```

   b. Describing data (counts, averages, min/max etc..)

```
df.describe().show()
```

▸ (2) Spark Jobs

```
+-------+------------------+------------------+------------------+
|summary|           movieId|            rating|            userId|
+-------+------------------+------------------+------------------+
|  count|              1501|              1501|              1501|
|   mean| 49.40572951365756|1.7741505662891406|14.383744170552964|
| stddev|28.937034065088994| 1.187276166124803| 8.591040424293272|
|    min|                 0|                 1|                 0|
|    max|                99|                 5|                29|
+-------+------------------+------------------+------------------+
```

   c. Distinct number of User Ids

```
df.select("userId").distinct().count()
```

▸ (3) Spark Jobs

```
Out[37]: 30
```

d. Distinct number of Movie Ids

```
df.select("movieId").distinct().count()
```

▸ (3) Spark Jobs

```
Out[38]: 100
```

e. Describing how many movies users have reviewed (average, min/max, etc..)

```
#calculating the number of times users reviewed movies
userrating_counts = df.groupBy("userId").count()
display(userrating_counts)
```

▸ (2) Spark Jobs

▸ userrating_counts: pyspark.sql.dataframe.DataFrame = [userId: integer, count: long]

	userId	count
1	12	55
2	13	48
3	14	57
4	18	52
5	25	46
6	6	57
7	9	53

Showing all 30 rows.

```
#viewing how many movies ppl have reviewed
userrating_counts.describe(['count']).show()
```

▸ (3) Spark Jobs

```
+-------+------------------+
|summary|             count|
+-------+------------------+
|  count|                30|
|   mean|  50.0333333333333|
| stddev|3.8460309441017904|
|    min|                44|
|    max|                57|
+-------+------------------+
```

f. Top 12 users and ratings

Top 12 highest ratings

```
#calculating top 12 movies with highest ratings
df.orderBy('rating', ascending=False).show(12)
```

▸ (1) Spark Jobs

```
+-------+------+------+
|movieId|rating|userId|
+-------+------+------+
|     55|     5|     5|
|     30|     5|    11|
|     25|     5|     6|
|      8|     5|     2|
|     25|     5|     7|
|     39|     5|     2|
|     29|     5|     8|
|     51|     5|     3|
|     52|     5|     8|
|      7|     5|     9|
|     27|     5|    11|
|     49|     5|     9|
+-------+------+------+
only showing top 12 rows
```

Top 12 users

```
#calculating top 12 highest number of ratings done by users
userrating_counts.orderBy('count', ascending=False).show(12)
```

▸ (2) Spark Jobs

```
+------+-----+
|userId|count|
+------+-----+
|     6|   57|
|    14|   57|
|    11|   56|
|    22|   56|
|     4|   55|
|    12|   55|
|     7|   54|
|     9|   53|
|    23|   52|
|    24|   52|
|    18|   52|
|    28|   50|
+------+-----+
only showing top 12 rows
```

2. Train-test Split
   a. Model 1:

```
#MODEL 1
#first combination split - 80/20
(train1, test1) = df.randomSplit([0.80, 0.20])
# Train model 1 with Training Data 1
als_model_1 = als.fit(train1)
#Model 1 predictions on test1
pred_1 = als_model_1.transform(test1)
```

▸ (5) Spark Jobs

▸ train1: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
▸ test1: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
▸ pred_1: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]

   b. Model 2

```
#MODEL 2
#second combination split - 60/40
(train2, test2) = df.randomSplit([0.6, 0.4])
# Train model 2 with Training Data 2
als_model_2 = als.fit(train2)
#Model 2 predictions on test1
pred_2 = als_model_2.transform(test2)
```

▸ (5) Spark Jobs

▸ train2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
▸ test2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]
▸ pred_2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]

   c. Initial ALS Model

```
#Create initial ALS model
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", coldStartStrategy="drop")
```

3. Report Models Performance
   a. MSE, RMSE, and MAE.

MSE stands for Mean Squared Error and represents the average of the squared difference between the actual and predicted values. RMSE stands for Root Mean Squared Error and measures the standard deviation of residuals by taking the square root of Mean Squared Error. MAE stands for Mean Absolute Error and represents the average of the absolute difference between the actual and predicted values. The smaller the value of MSE, RMSE and MAE, the better. A smaller value indicates better accuracy.

```
#EVALUATORS
#Evaluation with RMSE
evaluator_RMSE = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
#Evaluation with MSE
evaluator_MSE = RegressionEvaluator(metricName="mse", labelCol="rating", predictionCol="prediction")
#Evaluation with MAE
evaluator_MAE = RegressionEvaluator(metricName="mae", labelCol="rating", predictionCol="prediction")
```

b. Which one of the two splits worked better?

Model 1 works better as it has a lower RMSE, MSE and MAE than Model 2 indicating higher accuracy. Model 1 works better because of the higher training split. Model 1 trains the model on 80% of the data while Model 2 trains the model on only 60% of the data. However, larger test datasets provide a more accurate representation of the model's performance.

```
#MODEL 1 EVALUATION
#RMSE Model 1
rmse_1 = evaluator_RMSE.evaluate(pred_1)
#MSE Model 1
mse_1 = evaluator_MSE.evaluate(pred_1)
#MAE Model 1
mae_1 = evaluator_MAE.evaluate(pred_1)
```

▸ (15) Spark Jobs

```
#MODEL 2 EVALUATION
#RMSE Model 2
rmse_2 = evaluator_RMSE.evaluate(pred_2)
#MSE Model 2
mse_2 = evaluator_MSE.evaluate(pred_2)
#MAE Model 2
mae_2 = evaluator_MAE.evaluate(pred_2)
```

▸ (15) Spark Jobs

```
print ("RMSE of model 1: ", rmse_1, "\nRMSE of model 2: ", rmse_2)
print ("MSE of model 1: ", mse_1, "\nMSE of model 2: ", mse_2)
print ("MAE of model 1: ", mae_1, "\nMAE of model 2: ", mae_2)
```

```
RMSE of model 1:  1.0223004355696874
RMSE of model 2:  1.1692273433760312
MSE of model 1:  1.0450981805659725
MSE of model 2:  1.3670925804981717
MAE of model 1:  0.6846297040877752
MAE of model 2:  0.8261748726854657
```

4. Model Tuning

a. Combination selected:
Model 1, 80/20 split (however model 2 was also tuned and run as per assignment instructions given during lecture)

b. What are the hyperparameters of ALS?
The hyperparameters of ALS are rank, maxIter, regParam, numBlocks, and alpha.

c. Which hyperparameters did you tune?
I chose to tune the hyperparameters rank and regParam. Rank is the number of latent factors in the model and defaults to 10. This hyperparameter helps to group/categorize how similar and different movies are. I decided to test out the following different rank values: 10, 50, 100 and 200. regParam specifies the regularization parameter and defaults to 1.0. I decided to test out the following different values: 0.1,0.15,0.2,0.25.

```
# Create ParamGrid for Cross Validation
parameters = (ParamGridBuilder()
              .addGrid(als.rank, [10, 50, 100, 200])
              .addGrid(als.regParam, [.1, .15, 0.2, 0.25])
              .build())

#Build train validation split
tranvs = TrainValidationSplit(estimator=als, estimatorParamMaps=parameters, evaluator=evaluator_RMSE)
```

```
#BEST MODEL 1
#Run train validation split
tuned_model_1 = tranvs.fit(train1)
#Get best Model
best_model = tuned_model_1.bestModel
```

▸ (9) Spark Jobs

d. Best Model and train-test error.

The Best Model uses a rank of 200 and a regParam of 0.15 to get a RMSE of 0.994.

```
#Using best model to predict
tuned_pred_1 = best_model.transform(test1)
tuned_pred_2 = best_model2.transform(test2)
```

▸ tuned_pred_1: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
▸ tuned_pred_2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]

Command took 0.11 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

Cmd 66

```
#RMSE of Best Model 1
tuned_rmse_1 = evaluator_RMSE.evaluate(tuned_pred_1)
#RMSE of Best Model 2
tuned_rmse_2 = evaluator_RMSE.evaluate(tuned_pred_2)
```

▸ (10) Spark Jobs

Command took 7.37 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

Cmd 67

```
print ("RMSE of new model 1: ", tuned_rmse_1)
print ("RMSE of new model 2: ", tuned_rmse_2)
```

```
RMSE of new model 1:  0.9944893950996873
RMSE of new model 2:  1.1479176736055905
```

Command took 0.03 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 12:51:59 AM on Cluster_Assign2

5. Recommended Movies
   The Best Model (best_model) was used for this question.

```
# Generate Recommendations for all users (from best to worst)
movie_recommendations = best_model.recommendForAllUsers(100)
movie_recommendations.show()
```

▸ (2) Spark Jobs

▸ movie_recommendations: pyspark.sql.dataframe.DataFrame = [userId: integer, recommendations: array]

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|    28|[[22, 3.9048154],...|
|    10|[[92, 3.6125093],...|
|     8|[[92, 3.11789], [...|
|     1|[[62, 3.1660814],...|
```

a. Top 12 movies recommendations for User Id 1

```
#filtering for only User Id 1
movie_recommendations_ID1 = movie_recommendations.filter(movie_recommendations.userId == 1)
movie_recommendations_ID1.show()
```

▸ (2) Spark Jobs

▸ movie_recommendations_ID1: pyspark.sql.dataframe.DataFrame = [userId: integer, recommendations: array]

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|     1|[[62, 3.1660814],...|
+------+--------------------+
```

Command took 17.41 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 1:42:46 AM on Cluster_Assign2

Cmd 57

```
#User Id 1
#Creating a list of all the movies that user Id 1 reviewed
userID1movies = df.filter(col("userId") == 1).select('movieId').rdd.flatMap(lambda x: x).collect()

#Using explode to format the array into columns
#filtering out the movies that the user has already reviewed and showing top 12
recommendations_ID1 = movie_recommendations_ID1.withColumn("recommendations", explode("recommendations")).select('userId', col("recommendations.movieId"), col("recommendations.rating"))
recommendations_ID1.filter(~recommendations_ID1.movieId.isin(userID1movies)).show(12)
```

▸ (2) Spark Jobs

▸ recommendations_ID1: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 1 more field]

```
+------+-------+---------+
|userId|movieId|   rating|
+------+-------+---------+
|     1|     22|  2.30624|
|     1|      7| 2.273389|
|     1|     51| 2.232317|
|     1|     95|2.0818558|
|     1|     31|2.0718462|
|     1|     80|2.0705826|
|     1|     25|2.0507987|
|     1|     29|1.9632709|
|     1|     98|1.9516469|
|     1|     23|1.9421765|
|     1|     24|1.8715812|
|     1|     32|1.8000154|
+------+-------+---------+
only showing top 12 rows
```

b. Top 12 movie recommendations for User Id 12

```
#filtering for only User Id 12
movie_recommendations_ID12 = movie_recommendations.filter(movie_recommendations.userId == 12)
movie_recommendations_ID12.show()
```

▸ (2) Spark Jobs

▸ movie_recommendations_ID12: pyspark.sql.dataframe.DataFrame = [userId: integer, recommendations: array]

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|    12|[[55, 4.326746], ...|
+------+--------------------+
```

Command took 17.83 seconds -- by melina.fartaj@mail.utoronto.ca at 10/11/2021, 1:42:46 AM on Cluster_Assign2

Cmd 59

```
#User Id 12
#Creating a list of all the movies that user Id 12 reviewed
userID12movies = df.filter(col("userId") == 12).select('movieId').rdd.flatMap(lambda x: x).collect()

#Using explode to format the array into columns
#filtering out the movies that the user has already reviewed and showing top 12
recommendations_ID12 = movie_recommendations_ID12.withColumn("recommendations", explode("recommendations")).select('userId', col("recommendations.movieId"), col("recommendations.rating"))
recommendations_ID12.filter(~recommendations_ID12.movieId.isin(userID12movies)).show(12)
```

▸ (2) Spark Jobs

▸ recommendations_ID12: pyspark.sql.dataframe.DataFrame = [userId: integer, movieId: integer ... 1 more field]

```
+------+-------+---------+
|userId|movieId|   rating|
+------+-------+---------+
|    12|     55| 4.326746|
|    12|     46| 3.583644|
|    12|     49|3.5201106|
|    12|     65|3.4627342|
|    12|     48| 3.369004|
|    12|     20| 3.341531|
|    12|     32|3.2379308|
|    12|     90|3.1749177|
|    12|     10|2.9882762|
|    12|     68|2.9059153|
|    12|      1|2.8913836|
|    12|     28| 2.647625|
+------+-------+---------+
only showing top 12 rows
```