Part A

Q1 :

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = SparkSession.builder.appName("DataFrame").getOrCreate()

# File location and type
file_location = "/FileStore/tables/integer.txt"
df = spark.sparkContext.textFile(file_location)
arr= df.map(lambda x : int(x))
arr_data=arr.collect()
```

▸ (1) Spark Jobs

Command took 7.73 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 2:58:45 PM on Rutvik Solanki's Cluster

Cmd 3

Python

```python
odd,even=0,0
for i in arr_data:

    if i%2==0:
        even+=1

    else:

        odd+=1
print("Number of odd integers are :",odd)
print("Number of even integers are:",even)
```

```
Number of odd integers are : 496
Number of even integers are: 514
```

Command took 0.09 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 2:59:08 PM on Rutvik Solanki's Cluster

Cmd 4

Q2:

Cmd 2

Python

```python
#all spark imports
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = SparkSession.builder.appName("Q2").getOrCreate()
# File location and type
file_location = "/FileStore/tables/salary.txt"
df = spark.sparkContext.textFile("/FileStore/tables/salary.txt")

data=df.map(lambda x :(x.split(" ")))
data=data.map(lambda x:(x[0],int(x[1])))
data=data.reduceByKey(lambda x,y:x+y)
for i in data.collect():
    print(i)
```

▸ (1) Spark Jobs

```
('Sales', 3488491)
('Research', 3328284)
('Developer', 3221394)
('QA', 3360624)
('Marketing', 3158450)
```

Command took 2.92 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:02:50 PM on Rutvik Solanki's Cluster

Cmd 3

Q3

```
#all spark imports
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = SparkSession.builder.getOrCreate()
# File location and type
file_location = "/FileStore/tables/shakespeare_1.txt"
text= sc.textFile(file_location)
given_words="Shakespeare,Why,Lord,Library,GUTENBERG,WILLIAM,COLLEGE,WORLD"
given_words=given_words.split(",")
#print(given_words)

def clean_data(s):
    data=s.strip(string.punctuation)
    return(data)

df=text.map(clean_data)
#print(df.collect())
df=df.flatMap(lambda x: re.split(r"\W+",x))
#print(df.collect())
df=df.filter(lambda x : len(x)>1)
df=df.map(lambda word :(word,1))
df=df.reduceByKey(lambda a,b:a+b)
#print(df.collect())

for i in df.collect():
    if i[0] in given_words:
        print(i)
```

▸ (1) Spark Jobs

```
('Shakespeare', 22)
('GUTENBERG', 100)
('WILLIAM', 128)
('WORLD', 98)
('COLLEGE', 98)
('Lord', 402)
('Library', 5)
('Why', 494)
```

Command took 2.19 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:05:01 PM on Rutvik Solanki's Cluster

Q4:

Cmd 4

```
res=df.sortBy(lambda x : x[1]).collect()
print("bottom 15 words are:")
print(res[:15])
```

▸ (3) Spark Jobs

```
bottom 15 words are:
[('anyone', 1), ('restrictions', 1), ('online', 1), ('www', 1), ('gutenberg', 1), ('org', 1), ('COPYRIGHTED', 1), ('Details', 1), ('guidelines', 1), ('Postin
g', 1), ('2011', 1), ('EBook', 1), ('January', 1), ('1994', 1), ('Character', 1)]
```

Command took 1.13 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:06:56 PM on Rutvik Solanki's Cluster

Cmd 5

```
print("Top 15 words in ascending order are:")
print(res[-15:])
```

```
Top 15 words in ascending order are:
[('it', 3078), ('with', 3221), ('his', 3278), ('me', 3448), ('not', 3595), ('is', 3722), ('And', 3735), ('that', 3864), ('in', 4803), ('my', 4922), ('you', 536
0), ('to', 7742), ('of', 7968), ('and', 8942), ('the', 11412)]
```

Command took 0.08 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:06:56 PM on Rutvik Solanki's Cluster

Cmd 6

Part B :

Q1

i. Describing the dataset.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

spark = SparkSession.builder.getOrCreate()

# File location and type
file_location = "/FileStore/tables/movies.csv"
df = spark.read.csv(file_location,inferSchema=True,header=True)
#df.show()
df.describe().show()
```

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

+-------+------------------+------------------+------------------+
|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|
+-------+------------------+------------------+------------------+

Command took 10.62 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:07:59 PM on Rutvik Solanki's Cluster
```

ii. Top 20 movies with highest rating also sorting the movieID in ascending order.

Cmd 3

Python

```
#top 20 movies with highest rating.
#assuming movieId to be asceding order.
x=df.groupBy("movieId").count()
x.sort(col('count').desc(),col('movieId').asc()).show(20)
```

▸ (2) Spark Jobs

▸ x: pyspark.sql.dataframe.DataFrame = [movieId: integer, count: long]

```
+-------+-----+
|movieId|count|
+-------+-----+
|      6|   20|
|     22|   20|
|     29|   20|
|     50|   20|
|     51|   20|
|      2|   19|
|     15|   19|
|     55|   19|
|     68|   19|
|     94|   19|
|     14|   18|
|     36|   18|
|     45|   18|
|     85|   18|
|     86|   18|
|     88|   18|
|      4|   17|
|     12|   17|
```

Command took 0.49 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:14:14 PM on Rutvik Solanki's Cluster

Cmd 4

iii. Top 10 users who have given the highest rating. Assuming userID and movieID to be sorted in ascending order.

```python
#top 15 users who have given the highest rating.
#assuming UserId to be asc() order.
res=df.groupBy("userID").count()
res.sort(col('count').desc(),col('userID').asc()).show(15)

```

▸ (2) Spark Jobs

▸ res: pyspark.sql.dataframe.DataFrame = [userID: integer, count: long]

```
+------+-----+
|userID|count|
+------+-----+
|     6|   57|
|    14|   57|
|    11|   56|
|    22|   56|
|     4|   55|
|    12|   55|
|     7|   54|
|     9|   53|
|    18|   52|
|    23|   52|
|    24|   52|
|    28|   50|
|     0|   49|
|     1|   49|
|     5|   49|
+------+-----+
only showing top 15 rows
```

Command took 0.84 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:14:55 PM on Rutvik Solanki's Cluster

Q2.

```python
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS

(training_1, test_1) = df.randomSplit([0.8, 0.2])
(training_2, test_2) = df.randomSplit([0.7, 0.3])

als= ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",coldStartStrategy="drop")
model_1= als.fit(training_1)
#als_2=als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",coldStartStrategy="drop")
model_2= als.fit(training_2)
print("Model_1 output:")
print(model_1.transform(test_1).collect())
print("Model_2 output:")
print(model_2.transform(test_2).collect())


```

▸ (14) Spark Jobs

▸ training_1: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]

▸ test_1: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 1 more field]

▸ 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]

```
Model_1 output:
[Row(movieId=0, rating=1, userId=8, prediction=2.9000205993652344), Row(movieId=0, rating=1, userId=26, prediction=-0.22002136707305908), Row(movieId=1, rating
=1, userId=26, prediction=1.2154017686843872), Row(movieId=1, rating=4, userId=15, prediction=0.10710091888904572), Row(movieId=2, rating=1, userId=3, predicti
on=1.2413959503173828), Row(movieId=2, rating=1, userId=26, prediction=4.047211647033691), Row(movieId=2, rating=2, userId=20, prediction=2.218501091003418), R
ow(movieId=3, rating=1, userId=9, prediction=0.46156764030456543), Row(movieId=3, rating=2, userId=8, prediction=1.581174373626709), Row(movieId=4, rating=1, u
serId=7, prediction=0.7430272698402405), Row(movieId=4, rating=1, userId=12, prediction=-0.6524271368980408), Row(movieId=4, rating=1, userId=24, prediction=3.
1595003604888916), Row(movieId=5, rating=1, userId=9, prediction=2.560699701309204), Row(movieId=5, rating=3, userId=16, prediction=1.7469247579574585), Row(mo
vieId=6, rating=1, userId=15, prediction=1.583225965499878), Row(movieId=6, rating=2, userId=11, prediction=0.764802873134613), Row(movieId=6, rating=2, userId
=16, prediction=1.6938296556472778), Row(movieId=6, rating=3, userId=26, prediction=2.350095510482788), Row(movieId=8, rating=1, userId=5, prediction=2.6948685
64605713), Row(movieId=8, rating=1, userId=12, prediction=0.12047100067138672), Row(movieId=8, rating=1, userId=20, prediction=-1.9259867668151855), Row(movieI
d=9, rating=1, userId=14, prediction=2.551612377166748), Row(movieId=9, rating=3, userId=5, prediction=1.1263009309768677), Row(movieId=10, rating=1, userId=1
0, prediction=0.5582469701766968), Row(movieId=10, rating=4, userId=17, prediction=3.1022462844848633), Row(movieId=11, rating=1, userId=14, prediction=1.50113
09385299683), Row(movieId=11, rating=4, userId=18, prediction=0.35621878504753113), Row(movieId=12, rating=1, userId=23, prediction=-0.007511138916015625), Row
(movieId=12, rating=2, userId=0, prediction=2.7429275512695312), Row(movieId=12, rating=3, userId=2, prediction=0.32026612758636475), Row(movieId=13, rating=1,
userId=1, prediction=2.0442681312561035), Row(movieId=13, rating=3, userId=29, prediction=2.108077049255371), Row(movieId=14, rating=1, userId=28, prediction=
0.4520137906074524), Row(movieId=14, rating=2, userId=7, prediction=0.464892715215683), Row(movieId=14, rating=3, userId=21, prediction=2.3562369346618652), Ro
```

Q3.

Cmd 6

Python

```python
predictions_1 = model_1.transform(test_1)
evaluator_1= RegressionEvaluator(metricName="rmse", labelCol="rating",predictionCol="prediction")
rmse = evaluator_1.evaluate(predictions_1)
print("Root-mean-square error for model_1 =" + str(rmse))

predictions_2 = model_2.transform(test_2)
evaluator_2= RegressionEvaluator(metricName="rmse", labelCol="rating",predictionCol="prediction")
rmse = evaluator_2.evaluate(predictions_2)
print("Root-mean-square error for model_2=" + str(rmse))

predictions_3= model_1.transform(test_1)
evaluator_3= RegressionEvaluator(metricName="mae", labelCol="rating",predictionCol="prediction")
mae = evaluator_3.evaluate(predictions_3)
print("mean-absolute error for model_1 =" + str(mae))

predictions_4 = model_2.transform(test_2)
evaluator_4= RegressionEvaluator(metricName="mae", labelCol="rating",predictionCol="prediction")
mae = evaluator_4.evaluate(predictions_4)
print("mean-absolute error for model_2 =" + str(mae))
```

```
(21) Spark Jobs
predictions_1:  pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
predictions_2:  pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
predictions_3:  pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
predictions_4:  pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
Root-mean-square error for model_1 =1.7711159462919928
Root-mean-square error for model_2=1.7810416325329739
mean-absolute error for model_1 =1.3200413913489237
mean-absolute error for model_2 =1.3468710078444894

Command took 7.64 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:16:47 PM on Rutvik Solanki's Cluster
```

Answer: Numerous regression models utilize distance-based metrics to assess their performance. The fundamental concept is to evaluate how far the predicted values deviate from the original ones. There are three different error metrics commonly used: MAE, MSE, and RMSE.

MAE (Mean Absolute Error) calculates the average absolute difference between the predicted and original values.

MSE (Mean Squared Error) squares the difference between the predicted and original values, calculates the average, and provides a mean squared value.

RMSE (Root Mean Squared Error) takes the square root of the MSE, hence the term "Root" in its name.

It is important to note that these error metrics do not consider the direction of the error (positive or negative), and a lower error value indicates a better fit for the model. Now, the question arises: **which error metric is better?**

There is no definitive answer to this question as it depends on the dataset. RMSE penalizes larger differences more than MAE, while MAE provides a more general value (also, MAE is always less than or equal to RMSE due to their calculation methods). Typically, it is preferred to penalize large errors (outliers) to ensure a more generalized model. Therefore, for the remaining part of this assignment, we will use RMSE as the evaluation metric rather than MSE.

Now, let's analyze which metric works better and why. Looking at the RMSE results, we observe that Model 1 (80:20 ratio) has a lower value (1.7) compared to Model 2 (70:30 ratio with a value of 2.17). The reason for this lower error value is that Model 1 has a larger training set ratio. By including more data in the training process, the model learns the features better and can perform better on the test data. If we do not include sufficient training data, our model might not learn to handle outliers or data with high variance, which can limit its generalization capability. To further improve the training process, techniques like K-fold cross-validation can be employed.

Q4.

Python

```
from pyspark.ml.tuning import *

rank_val=[1,2,3,4,5,6,7,8,9,10]
iter_val=[1,2,3,4,5,6,7,8,9,10]
regParam_val=[0.001,0.1,0.2,0.002,0.003,0.3,0.004,0.4,0.005,0.5]
res_1,res_2=[],[]
for i in range(10):
    als= ALS(rank=rank_val[i],maxIter=iter_val[i], regParam=regParam_val[i], userCol="userId", itemCol="movieId", ratingCol="rating",
    coldStartStrategy="drop")
    model_1=als.fit(training_1)
    model_2=als.fit(training_2)
    p_1=model_1.transform(test_1)
    p_2=model_2.transform(test_2)
    evaluator= RegressionEvaluator(metricName="rmse", labelCol="rating",predictionCol="prediction")
    rmse_1=evaluator.evaluate(p_1)
    rmse_2=evaluator.evaluate(p_2)
    res_1.append(float(rmse_1))
    res_2.append(float(rmse_2))


print("Model 1 errors are:", res_1)
print("Model 2 error are", res_2)

```

(4) Spark Jobs

p_1: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]

p_2: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]

```
Model 1 errors are: [1.9527578952561695, 1.2037644777981653, 1.1280374131217124, 1.7948152870328165, 1.6931351670437922, 1.0470989071852763, 1.9229005048261794, 1.1166975116724804, 1.7836362891068234, 1.1869479177610354]
Model 2 error are [1.9358018424211836, 1.2902578093679653, 1.241565365660664, 1.8675203437395873, 1.987242812943077, 1.103883421214468, 2.2008710991250093, 1.1675183004267358, 2.2499466837657716, 1.2276930238868733]
```

Command took 2.62 minutes -- by rutvikrj26@gmail.com at 6/22/2023, 3:18:48 PM on Rutvik Solanki's Cluster

Cmd 8

```
import builtins as p
print("The best rmse error for model 1 is :",p.min(res_1))
idx=res_1.index(p.min(res_1))
#print(idx)
print("The following the values are of the model:")
print("Rank:",rank_val[idx],"Iteration :",iter_val[idx],"RegParam :",regParam_val[idx])
print("The best rmse error for model 2 is :",p.min(res_2))
idx=res_2.index(p.min(res_2))
#print(idx)
print("The following the values are of the model:")
print("Rank:",rank_val[idx],"Iteration :",iter_val[idx],"RegParam :",regParam_val[idx])

```

```
The best rmse error for model 1 is : 1.0470989071852763
The following the values are of the model:
Rank: 6 Iteration : 6 RegParam : 0.3
The best rmse error for model 2 is : 1.103883421214468
The following the values are of the model:
Rank: 6 Iteration : 6 RegParam : 0.3
```

Command took 0.07 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:23:40 PM on Rutvik Solanki's Cluster

Cmd 9

Hyperparameter tuning

From the official documentation (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.recommendation.ALS.html?highlight=als#pyspark.ml.recommendation.ALS) we can see the following parameters can be optimized

ALS

class pyspark.ml.recommendation.ALS(**, rank=10, maxIter=10, regParam=0.1, numUserBlocks=10, numItemBlocks=10, implicitPrefs=False, alpha=1.0, userCol='user', itemCol='item', seed=None, ratingCol='rating', nonnegative=False, checkpointInterval=10, intermediateStorageLevel='MEMORY_AND_DISK', finalStorageLevel='MEMORY_AND_DISK', coldStartStrategy='nan', blockSize=4096*) [source]

Alternating Least Squares (ALS) matrix factorization.

1. Rank – number of latent factors in model
2. maxIter- the number of iterations go back and forth between the two matrix.
3. RegParam- regularization for parameter in ALS
4. alpha- baseline confidence
5. numUserBlocks- user defined memory blocks for parallelization.

Note :- In Databricks community edition, limited compute power was utilized. The computation is extremely slow to optimize all the parameters and hyperparameter tuning in this case is a iterative process, hence we might not to able to find the best model given the computation resources allocated . However, I ran 10 different models with small changes and results of the best model is presented in this case.

Q5.

Cmd 9

```
als_best= ALS(rank=6,maxIter=6, regParam=0.3, userCol="userId", itemCol="movieId", ratingCol="rating",coldStartStrategy="drop")
best_model=als_best.fit(training_1)
predicitons=best_model.transform(test_1)
user_recs=best_model.recommendForAllUsers(15)
```

```
(5) Spark Jobs
predicitons: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: integer ... 2 more fields]
user_recs: pyspark.sql.dataframe.DataFrame = [userId: integer, recommendations: array]
Command took 5.85 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:27:19 PM on Rutvik Solanki's Cluster
```

Cmd 10

```
data=user_recs.filter((user_recs.userId==10)|(user_recs.userId==14))
data.show()
```

```
(2) Spark Jobs
data: pyspark.sql.dataframe.DataFrame = [userId: integer, recommendations: array]
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|    10|[{2, 2.2106357}, ...|
|    14|[{29, 3.122271}, ...|
+------+--------------------+

Command took 7.42 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:27:19 PM on Rutvik Solanki's Cluster
```

Extracting and presenting the data in a readable format.

Cmd 11

```
#Extracted data for UserId==10
data.select(col("recommendations")).rdd.flatMap(list).collect()[0]
```

(2) Spark Jobs

```
Out[26]: [Row(movieId=2, rating=2.2106356620788574),
 Row(movieId=25, rating=2.174492835998535),
 Row(movieId=89, rating=2.150029420852661),
 Row(movieId=62, rating=1.975149154663086),
 Row(movieId=49, rating=1.8908214569091797),
 Row(movieId=87, rating=1.7991971969604492),
 Row(movieId=93, rating=1.761389136314392),
 Row(movieId=46, rating=1.7298362255096436),
 Row(movieId=32, rating=1.7290337085723877),
 Row(movieId=92, rating=1.6943535804748535),
 Row(movieId=58, rating=1.6560285091400146),
 Row(movieId=91, rating=1.645521879196167),
 Row(movieId=81, rating=1.6438124179840088),
 Row(movieId=40, rating=1.6317248344421387),
 Row(movieId=53, rating=1.6281766891479492)]
```

Command took 7.02 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:27:23 PM on Rutvik Solanki's Cluster

Cmd 12

```
#Extracted data for UserId==14
data.select(col("recommendations")).rdd.flatMap(list).collect()[1]
```

(2) Spark Jobs

```
Out[27]: [Row(movieId=29, rating=3.1222710609436035),
 Row(movieId=85, rating=2.865478754043579),
 Row(movieId=25, rating=2.8615880012512207),
 Row(movieId=63, rating=2.8059139251708984),
 Row(movieId=53, rating=2.749671220779419),
 Row(movieId=62, rating=2.653148651123047),
 Row(movieId=52, rating=2.636749505996704),
 Row(movieId=58, rating=2.550471782684326),
 Row(movieId=76, rating=2.53499698638916),
 Row(movieId=2, rating=2.3991386890411377),
 Row(movieId=96, rating=2.383424758911133),
 Row(movieId=72, rating=2.342783212661743),
 Row(movieId=93, rating=2.268589735031128),
 Row(movieId=43, rating=2.266918659210205),
 Row(movieId=74, rating=2.1783649921417236)]
```

Command took 6.84 seconds -- by rutvikrj26@gmail.com at 6/22/2023, 3:27:27 PM on Rutvik Solanki's Cluster

TA can refer html files/ipynb notebooks given as attachments if required.