Before implementing the A2.ipynb, download spark to the specified directory. Install find spark in anaconda, and use jupyternote to implement the A2.ipynb.

In [1]:
```
import findspark
findspark.init('C:\spark-3.3.0-bin-hadoop3')
import findspark
findspark.find()
```

Out[1]:
```
'C:\\spark-3.3.0-bin-hadoop3'
```

Count the odd and even numbers

In [2]:
```
#all spark imports
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

#instantiate the spark session
spark = SparkSession.builder.appName("Titanic-Survival-Prediction").getOrCreate()

#set the shuffle partition same as number of cpu cores to improve performance
spark.conf.set("spark.sql.shuffle.partitions", 4)
```

In [3]:
```
from pyspark.ml.feature import Imputer, StringIndexer, VectorAssembler
from pyspark.ml.linalg import SparseVector, DenseVector
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

Part A

1. Count the odd and even numbers using file ‘integer.txt’ download it from the Quercus. Show your code and output.

   Answer: 496 odd numbers, 514 even numbers

:
```
from pyspark.sql.functions import col
df.groupBy((col("_c0")%2).alias("isOdd")).count().show()
```

```
+-----+-----+
|isOdd|count|
+-----+-----+
|    1|  496|
|    0|  514|
+-----+-----+
```

2. 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.

```
In [35]: df2.groupBy('_c0').sum('_c1').show()
```

```
+---------+--------+
|      _c0|sum(_c1)|
+---------+--------+
|    Sales| 3488491|
|Developer| 3221394|
|       QA| 3360624|
|Marketing| 3158450|
| Research| 3328284|
+---------+--------+
```

3. 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, why, Lord, Library, GUTENBERG, WILLIAM, COLLEGE and WORLD. (Count exact words only (marks will be deducted for incorrect lowercase/uppercase))

```
In [10]: import re
shakeRDD = spark.sparkContext.textFile("shakespeare-2.txt", 4)
times = shakeRDD.flatMap(lambda l: re.split(r'\W+',l))\
                .map(lambda w: (w,1))\
                .reduceByKey(lambda x, y: x + y)\
                .filter(lambda t: t[0] in ["Shakespeare","Why", "Lord", "Library", "GUTENBERG", "WILLIAM",
                                           "COLLEGE", "WORLD"])
```

```
In [11]: times.collect()
```

```
Out[11]: [('Shakespeare', 22),
 ('WILLIAM', 128),
 ('WORLD', 98),
 ('GUTENBERG', 100),
 ('COLLEGE', 98),
 ('Lord', 402),
 ('Library', 5),
 ('Why', 494)]
```

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

In [12]:
```
shakeRDD = spark.sparkContext.textFile("shakespeare-2.txt", 4)
words = shakeRDD.flatMap(lambda l: re.split(r'\W+',l))\
                .map(lambda w: (w,1))\
                .reduceByKey(lambda x, y: x + y)\
                .filter(lambda t: t[0] != '')\
                .sortBy(lambda d: -d[1])
words.collect()[:10]
```

Out[12]:
```
[('the', 11412),
 ('I', 9714),
 ('and', 8942),
 ('of', 7968),
 ('to', 7742),
 ('a', 5796),
 ('you', 5360),
 ('my', 4922),
 ('in', 4803),
 ('d', 4365)]
```

In [13]:
```
shakeRDD = spark.sparkContext.textFile("shakespeare-2.txt", 4)
words = shakeRDD.flatMap(lambda l: re.split(r'\W+',l))\
                .map(lambda w: (w,1))\
                .reduceByKey(lambda x, y: x + y)\
                .filter(lambda t: t[0] != '')\
                .sortBy(lambda d: d[1])
words.collect()[:10]
```

Out[13]:
```
[('www', 1),
 ('gutenberg', 1),
 ('Posting', 1),
 ('EBook', 1),
 ('Character', 1),
 ('encoding', 1),
 ('cooperation', 1),
 ('SHAREWARE', 1),
 ('PUBLIC', 1),
 ('DOMAIN', 1)]
```

Part B

1. Describe your data. Calculate top 20 movies with highest ratings and top 15 users who provided highest ratings. Show your code and output.

top 20 movies with highest ratings top 20 movies with highest ratings

```
In [22]: spark.conf.set("spark.sql.shuffle.partitions", 4)
```

```
In [23]: path = "movies.csv"

df = spark.read \
  .format("csv") \
  .option("inferSchema", True) \
  .option("header", True) \
  .option("sep", ',') \
  .option("path", path) \
  .load()
```

```
In [24]: df.show()
```

```
+-------+------+------+
|movieId|rating|userId|
+-------+------+------+
|      2|     3|     0|
|      3|     1|     0|
|      5|     2|     0|
|      9|     4|     0|
|     11|     1|     0|
|     12|     2|     0|
|     15|     1|     0|
|     17|     1|     0|
|     19|     1|     0|
|     21|     1|     0|
|     23|     1|     0|
|     26|     3|     0|
|     27|     1|     0|
|     28|     1|     0|
|     29|     1|     0|
|     30|     1|     0|
|     31|     1|     0|
|     34|     1|     0|
|     37|     1|     0|
|     41|     2|     0|
+-------+------+------+
only showing top 20 rows
```

Top 20 movies with highest ratings

```
In [25]: df.groupby("movieID").avg('rating').sort("avg(rating)", ascending = False).show(20)
```

```
+-------+------------------+
|movieID|       avg(rating)|
+-------+------------------+
|     32|2.9166666666666665|
|     90|            2.8125|
|     30|               2.5|
|     94| 2.473684210526316|
|     23| 2.466666666666667|
|     49|            2.4375|
|     18|               2.4|
|     29|               2.4|
|     52| 2.357142857142857|
|     62|              2.25|
|     53|              2.25|
|     92|2.2142857142857144|
|     46|               2.2|
|     68|2.1578947368421053|
|     87|2.1333333333333333|
|      2|2.1052631578947367|
|     69| 2.076923076923077|
|     27| 2.066666666666667|
|     88|2.0555555555555554|
|     22|              2.05|
+-------+------------------+
only showing top 20 rows
```

Top 15 users who provided highest ratings

```
In [26]: df.groupby("userId").avg('rating').sort("avg(rating)", ascending = False).show(15)
```

```
+------+------------------+
|userId|       avg(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|
|     9|1.7924528301886793|
|    14|1.7894736842105263|
|    16|1.7777777777777777|
+------+------------------+
only showing top 15 rows
```

2. 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.

Split dataset into train and test. (80,20 and 70,30) use collaborative filtering approach on 70 percent of your data and test with the other 30 percent

In [27]:
```
train1, test1 = df.randomSplit([0.7, 0.3])

train2, test2 = df.randomSplit([0.8, 0.2])


alsdata = ALS(userCol = "userId", itemCol = "movieId", ratingCol = "rating",
              coldStartStrategy = "drop")
model1 = alsdata.fit(train1)
model2 = alsdata.fit(train2)
```

3. 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?

   Mean squared error (MSE) measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values.

   Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are.

   mean absolute error (MAE) is a measure of errors between paired observations expressing the same phenomenon. Examples of Y versus X include comparisons of predicted versus observed, subsequent time versus initial time, and one technique of measurement versus an alternative technique of measurement. MAE is calculated as the sum of absolute errors divided by the sample size

In [29]:
```
eva_MSE = RegressionEvaluator(metricName = "mse", predictionCol="prediction", labelCol="rating")
pre1 = model1.transform(test1)
pre2 = model2.transform(test2)

mse1 = eva_MSE.evaluate(pre1)
mse2 = eva_MSE.evaluate(pre2)
print(mse1)
print(mse2)
```

```
1.1136784395132724
1.0176715586526854
```

In [30]:
```
eva_RMSE = RegressionEvaluator(metricName = "rmse", predictionCol="prediction", labelCol="rating")

rmse1 = eva_RMSE.evaluate(pre1)
rmse2 = eva_RMSE.evaluate(pre2)
print(rmse1)
print(rmse2)
```

```
1.0553096415333618
1.00879708497432
```

In [31]:
```
eva_MAE = RegressionEvaluator(metricName = "mae", predictionCol="prediction", labelCol="rating")

mae1 = eva_MAE.evaluate(pre1)
mae2 = eva_MAE.evaluate(pre2)
print(mae1)
print(mae2)
```

```
0.7575173862548971
0.703591212059242
```

From above results, the RMSE and MSE of model1(70/30) are similar with model2(80/20), but the MAE of model1 is higher, which means the mean of the absolute value of the errors for model1 is higher, therefore model2(80/20) is better.

4. 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.

   Tune the regParam and maxIter parameters and evaluate the models, the result is shown below.

In [33]:
```
para = ParamGridBuilder()\
        .addGrid(alsdata.regParam, [0.1, 1.0])\
        .addGrid(alsdata.maxIter, [10, 15])\
        .build()

CV = CrossValidator(estimator = alsdata, estimatorParamMaps = para, evaluator = eva_MSE, numFolds = 7)

model1 = CV.fit(train1)
best1 = model1.bestModel
pre1 = best1.transform(test1)

mse1 = eva_MSE.evaluate(pre1)
print(mse1)
```

```
1.0459846864560232
```

In [34]:
```
model2 = CV.fit(train2)
best2 = model2.bestModel
pre2 = best2.transform(test2)

mse2 = eva_MSE.evaluate(pre2)
print(mse2)
```

```
1.0224152006252087
```

The best MSE value for model1 is 1.0459846864560232, the best MSE value for model 2 is 1.0224152006252087.

5. Calculate top 15 movies recommendations for user id 10 and user id 14. Show your code and output.

In [35]:
```
recom = best1.recommendForAllUsers(20)
n = recom.withColumn("rec_exp", explode("recommendations"))\
         .select("userId", col("rec_exp.movieId"), col("rec_exp.rating"))
user10 = df.filter(df.userId == 10).select("movieId").rdd.flatMap(lambda x: x).collect()
n.filter(n.userId == 10).filter(n.movieId.isin(user10) == False).show(15)
```

```
+------+-------+---------+
|userId|movieId|   rating|
+------+-------+---------+
|    10|     92| 2.624387|
|    10|     12|2.5440228|
|    10|      9|2.2091775|
|    10|     62|2.2039583|
|    10|     81|2.1534595|
|    10|     91| 2.139664|
|    10|     46|1.9177157|
|    10|     71|1.8875595|
|    10|     82|1.7437273|
|    10|     88|1.7228769|
+------+-------+---------+
```

In [36]:
```
user14 = df.filter(df.userId == 14).select("movieId").rdd.flatMap(lambda x: x).collect()
n.filter(n.userId == 14).filter(n.movieId.isin(user14) == False).show(15)
```

```
+------+-------+---------+
|userId|movieId|   rating|
+------+-------+---------+
|    14|     85|3.3395948|
|    14|     58| 3.142126|
|    14|     43| 2.784018|
|    14|      2|2.6485991|
|    14|     60|2.4759626|
|    14|     74|2.4675732|
|    14|     41| 2.311631|
|    14|     77|2.2214978|
+------+-------+---------+
```