CIND719-DK0T
Final Exam Report:

Ramello Peralta 500519802

Question 1 People.csv and Relations.csv

Load the files into spark to create a GraphFrame

```
>>> from graphframes import *
>>> v = spark.read.schema('id string').format('csv').option('header','false').load
('/home/cu/FinalExam/People.csv')
```

- Vertices df must contain id column

```
>>> e = spark.read.schema('src string, dst string').format('csv').option('header',
'true').load('/home/cu/FinalExam/Relations.csv')
```

- Edges df must contain source and destination columns

```
>>> type(v)
<class 'pyspark.sql.dataframe.DataFrame'>
>>> type(e)
<class 'pyspark.sql.dataframe.DataFrame'>
```

```
>>> g = GraphFrame(v, e)
```

```
>>> type(g)
<class 'graphframes.graphframe.GraphFrame'>
```

Display the edges and vertices of the graph in Spark. 2 Pts

```
>>> g = GraphFrame(v, e)
>>> g.vertices.show()
+---+
| id|
+---+
|  A|
|  B|
|  C|
|  D|
|  E|
|  F|
|  G|
|  H|
|  I|
|  J|
|  K|
|  L|
|  M|
+---+
```

```
>>> g.edges.show()
+---+---+
|src|dst|
+---+---+
|  A|  B|
|  B|  C|
|  C|  A|
|  D|  A|
|  D|  B|
|  D|  E|
|  E|  B|
|  F|  E|
|  G|  A|
|  H|  B|
|  I|  B|
|  I|  H|
|  I|  M|
|  J|  I|
|  J|  K|
|  J|  M|
|  K|  H|
|  K|  L|
|  L|  J|
+---+---+
```

Display the degrees of each node in descending order. (e.g. Node B shall be first in the list of nodes as it has highest degree). 2 Pts

```
>>> from pyspark.sql.functions import *
>>> g.inDegrees.select('id','inDegree').orderBy(desc('inDegree')).show()
+---+--------+
| id|inDegree|
+---+--------+
|  B|       5|
|  A|       3|
|  H|       2|
|  M|       2|
|  E|       2|
|  L|       1|
|  J|       1|
|  C|       1|
|  K|       1|
|  I|       1|
+---+--------+
```

Extract nodes that are connected in circular-triangular patterns. (e.g. JKLJ) 4 Pts

```
>>> motif = g.find('(a)-[ab]->(b); (b)-[bc]->(c);(c)-[ca]->(a)')
>>> motif.show()
+---+------+---+------+---+------+
|  a|    ab|  b|    bc|  c|    ca|
+---+------+---+------+---+------+
|[A]|[A, B]|[B]|[B, C]|[C]|[C, A]|
|[B]|[B, C]|[C]|[C, A]|[A]|[A, B]|
|[C]|[C, A]|[A]|[A, B]|[B]|[B, C]|
|[J]|[J, K]|[K]|[K, L]|[L]|[L, J]|
|[K]|[K, L]|[L]|[L, J]|[J]|[J, K]|
|[L]|[L, J]|[J]|[J, K]|[K]|[K, L]|
+---+------+---+------+---+------+
```

- ABCA and JKLJ are cyclic

Question 2: BankChurners.csv

Load the BankChurners.csv file as a DataFrame show first 10 rows. 2 Pts

```
>>> df_bank = spark.read.format('csv').option('delimiter',',').option('quote','"')
.option('header','true').option('inferSchema','true').load('/home/cu/FinalExam/Ban
kChurners.csv')
```

- Schema is too large to type manually; inferred instead

```
>>> df_bank = df_bank.withColumnRenamed('Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1','NB1')
>>> df_bank = df_bank.withColumnRenamed('Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2','NB2')
```

```
>>> df_bank.show(10)
+---------+-----------------+------------+------+---------------+---------------+---------------+---------------+-----------------+--------+-------+
|CLIENTNUM|   Attrition_Flag|Customer_Age|Gender|Dependent_count|Education_Level|Marital_Status|Income_Category|Card_Category|Months_on_book|Total_Relationship_Count|Months_Inactive_12_mon|Contacts_Count_12_mon|Credit_Limit|Total_Revolving_Bal|Avg_Open_To_Buy|Total_Amt_Chng_Q4_Q1|Total_Trans_Amt|Total_Trans_Ct|Total_Ct_Chng_Q4_Q1|Avg_Utilization_Ratio|      NB1|    NB2|
+---------+-----------------+------------+------+---------------+---------------+---------------+---------------+-----------------+--------+-------+
|768805383|Existing Customer|          45|     M|              3|    High School|        Married|    $60K - $80K|             Blue|          39|                      5|                     1|                    3|     12691.0|              777|        11914.0|               1.335|           1144|            42|              1.625|                0.061|9.3448E-5|0.99991|
|818770008|Existing Customer|          49|     F|              5|       Graduate|         Single| Less than $40K|             Blue|          44|                      6|                     1|                    2|      8256.0|              864|         7392.0|               1.541|           1291|            33|              3.714|                0.105|5.6861E-5|0.99994|
|713982108|Existing Customer|          51|     M|              3|       Graduate|        Married|   $80K - $120K|             Blue|          36|                      4|                     1|                    0|      3418.0|                0|         3418.0|               2.594|           1887|            20|              2.333|                  0.0|2.1081E-5|0.99998|
|769911858|Existing Customer|          40|     F|              4|    High School|        Unknown| Less than $40K|             Blue|          34|                      3|                     4|                    1|      3313.0|             2517|          796.0|               1.405|           1171|            20|              2.333|                0.76|1.3366E-4|0.99987|
|709106358|Existing Customer|          40|     M|              3|     Uneducated|        Married|    $60K - $80K|             Blue|          21|                      5|                     1|                    0|      4716.0|                0|         4716.0|               2.175|            816|            28|              2.5|                  0.0|2.1676E-5|0.99998|
|713061558|Existing Customer|          44|     M|              2|       Graduate|        Married|    $40K - $60K|             Blue|          36|                      3|                     1|                    2|      4010.0|             1247|         2763.0|               1.376|           1088|            24|              0.846|                0.311|5.5077E-5|0.99994|
|810347208|Existing Customer|          51|     M|              4|        Unknown|        Married|        $120K +|             Gold|          46|                      6|                     1|                    3|     34516.0|             2264|        32252.0|               1.975|           1330|            31|              0.722|                0.066|1.2303E-4|0.99988|
|818906208|Existing Customer|          32|     M|              0|    High School|        Unknown|    $60K - $80K|           Silver|          27|                      2|                     2|                    2|     29081.0|             1396|        27685.0|               2.204|           1538|            36|              0.714|                0.048|8.5795E-5|0.99991|
|710930508|Existing Customer|          37|     M|              3|     Uneducated|         Single|    $60K - $80K|             Blue|          36|                      5|                     2|                    0|     22352.0|             2517|        19835.0|               3.355|           1350|            24|              1.182|                0.113|4.4796E-5|0.99996|
|719661558|Existing Customer|          48|     M|              2|       Graduate|         Single|   $80K - $120K|             Blue|          36|                      6|                     3|                    3|     11656.0|             1677|         9979.0|               1.524|           1441|            32|              0.882|                0.144|3.0251E-4| 0.9997|
+---------+-----------------+------------+------+---------------+---------------+---------------+---------------+-----------------+--------+-------+
only showing top 10 rows
```

- Long names of columns affected visibility--last two Naive Bayes columns were renamed

Get the average transaction amounts (“Total_Trans_Amt”) for different Marital_Status and Gender values. For example, average transaction amount of married and male customers is around 4244.4382$. Display the output. 4 Pts.

```
>>> df_bank.registerTempTable('Bank_churn_data')
>>> avg_transactions = spark.sql("select gender, marital_status, avg(total_trans_amt) as avg_transactions
 from Bank_churn_data group by gender, marital_status order by avg_transactions desc")
>>> avg_transactions.show(10)
+------+--------------+------------------+
|gender|marital_status| avg_transactions|
+------+--------------+------------------+
|     M|       Unknown|4757.550135501355|
|     M|        Single|4741.208470847085|
|     F|       Unknown|4683.547368421053|
|     F|      Divorced|4534.557213930349|
|     M|      Divorced|4522.739884393063|
|     F|        Single|4469.087529411765|
|     M|       Married|4244.438282647585|
|     F|       Married|4108.627498980009|
+------+--------------+------------------+
```

Get the max transaction amounts (“Total_Trans_Amt”) for different Income Categories. The resulting information will have two columns. 4 Pts.

- Income_category and max(total_trans_amt)

```
>>> max_transactions = spark.sql("select income_category, max(total_trans_amt) as max_transactions from b
ank_churn_data group by income_category order by max_transactions")
>>> max_transactions.show()
+---------------+----------------+
|income_category|max_transactions|
+---------------+----------------+
|        $120K +|           16695|
|        Unknown|           16908|
|   $80K - $120K|           17498|
| Less than $40K|           17628|
|    $40K - $60K|           17744|
|    $60K - $80K|           18484|
+---------------+----------------+
```

Question 3 Pima Indians dataset

Load the dataset in Spark and display 10 rows. 1 Pts

```
>>> df_pima = spark.read.format('csv').option('delimiter',',').option('header','true').option('inferSchem
a','true').load('/home/cu/FinalExam/pimaindians.csv')
>>> df_pima.show(10)
+---+-------+------+-------+-------+----+-----+---+-----+
|prg|glucose|bldprs|triceps|insulin| bmi| diab|age|class|
+---+-------+------+-------+-------+----+-----+---+-----+
|  6|    148|    72|     35|      0|33.6|0.627| 50|    1|
|  1|     85|    66|     29|      0|26.6|0.351| 31|    0|
|  8|    183|    64|      0|      0|23.3|0.672| 32|    1|
|  1|     89|    66|     23|     94|28.1|0.167| 21|    0|
|  0|    137|    40|     35|    168|43.1|2.288| 33|    1|
|  5|    116|    74|      0|      0|25.6|0.201| 30|    0|
|  3|     78|    50|     32|     88|31.0|0.248| 26|    1|
| 10|    115|     0|      0|      0|35.3|0.134| 29|    0|
|  2|    197|    70|     45|    543|30.5|0.158| 53|    1|
|  8|    125|    96|      0|      0| 0.0|0.232| 54|    1|
+---+-------+------+-------+-------+----+-----+---+-----+
only showing top 10 rows
```

Use appropriate method (such as VectorAssembler) to prepare features for the dataset. If you get stuck on this step, then you can use pimaindians_x.csv and move on to the next task. 4 Pts

```
>>> from pyspark.ml.feature import VectorAssembler
>>> assembler = VectorAssembler(inputCols=['prg','glucose','bldprs','triceps','insulin','bmi','diab','age
'], outputCol='features')
>>> df_pimax = assembler.transform(df_pima)
>>> df_pimax.show(10)
+---+-------+------+-------+-------+----+-----+---+-----+--------------------+
|prg|glucose|bldprs|triceps|insulin| bmi| diab|age|class|            features|
+---+-------+------+-------+-------+----+-----+---+-----+--------------------+
|  6|    148|    72|     35|      0|33.6|0.627| 50|    1|[6.0,148.0,72.0,3...|
|  1|     85|    66|     29|      0|26.6|0.351| 31|    0|[1.0,85.0,66.0,29...|
|  8|    183|    64|      0|      0|23.3|0.672| 32|    1|[8.0,183.0,64.0,0...|
|  1|     89|    66|     23|     94|28.1|0.167| 21|    0|[1.0,89.0,66.0,23...|
|  0|    137|    40|     35|    168|43.1|2.288| 33|    1|[0.0,137.0,40.0,3...|
|  5|    116|    74|      0|      0|25.6|0.201| 30|    0|[5.0,116.0,74.0,0...|
|  3|     78|    50|     32|     88|31.0|0.248| 26|    1|[3.0,78.0,50.0,32...|
| 10|    115|     0|      0|      0|35.3|0.134| 29|    0|[10.0,115.0,0.0,0...|
|  2|    197|    70|     45|    543|30.5|0.158| 53|    1|[2.0,197.0,70.0,4...|
|  8|    125|    96|      0|      0| 0.0|0.232| 54|    1|[8.0,125.0,96.0,0...|
+---+-------+------+-------+-------+----+-----+---+-----+--------------------+
only showing top 10 rows
```

Split the dataset into 20% test and 80% training data. Set the seed as 123. 2 Pts

```
>>> train, test = df_pimax.randomSplit(weights = [0.80, 0.20], seed = 123)
```

Train a LogisticRegression model using training data. 2 Pts

```
>>> from pyspark.ml.classification import LogisticRegression
```

```
>>> lrmodel = LogisticRegression(featuresCol='features',labelCol='class').fit(train)
```

Test the model using test data and report confusion matrix. 3 Pts

```
>>> preds = lrmodel.transform(test)
```

```
>>> y_actual = preds.select(['class']).collect()
>>> y_preds = preds.select(['prediction']).collect()
```

```
>>> from sklearn.metrics import classification_report, confusion_matrix
```

```
>>> print(classification_report(y_actual, y_preds))
              precision    recall  f1-score   support

           0       0.85      0.91      0.88       108
           1       0.76      0.65      0.70        48

    accuracy                           0.83       156
   macro avg       0.80      0.78      0.79       156
weighted avg       0.82      0.83      0.82       156
```

```
>>> print(confusion_matrix(y_actual, y_preds))
[[98 10]
 [17 31]]
```

Print the model coefficients on the console. 3 Pts

```
>>> import pandas as pd
>>> listcols = df_pimax.columns.tolist()
```

```
>>> for i in range(len(listcols)-2):
...     print(f'Coefficient for {listcols[i]}: {lrmodel.coefficients[i]}')
...
Coefficient for prg: 0.10834653021094999
Coefficient for glucose: 0.039331939340025552
Coefficient for bldprs: -0.013309347017571771
Coefficient for triceps: 0.001352514078101943
Coefficient for insulin: -0.0019227468935531858
Coefficient for bmi: 0.08401608436057037
Coefficient for diab: 0.7742225404483769
Coefficient for age: 0.011645342675498755
```

End of report.