Welcome to Week 3 of the Big Data Capstone!

This document provides a running example of completing the Week 3 assignment :

- A shorter version with fewer comments is available as script: sparkMLlibClustering.py
- To run these commands in Cloudera VM: first run the setup script: setupWeek4.sh
- You can then copy paste these commands in pySpark.
- To open pySpark, refer to : Week 2 and Week 4 of the Machine Learning course
- Note that your dataset may be different from what is used here, so your results may not match with those shown here

```
In [2]: import pandas as pd from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
```

Import Data

First let us read the contents of the file ad-clicks.csv. The following commands read in the CSV file in a table format and removes any extra whitespaces. So, if the CSV contained ' userid ' it becomes 'userid'.

Note: that you must change the path to ad-clicks.csv to the location on your machine, if you want to run this command on your machine.

```
In [3]: adclicksDF =
pd.read_csv('/Users/aloksingh/Downloads/capstone/second/courseraDataSimulation/ad-clicks.csv')
```

```
adclicksDF = adclicksDF.rename(columns=lambda x: x.strip()) #remove whitespaces from headers
```

Display Data

Let us display the first 5 lines of adclicksDF:

```
In [4]:adclicksDF.head(n=5)
```

Out[4]:

	timestamp	txID	userSessionid	teamid	userid	adID	adCategory
0	2016-05-23 15:52:24	6045	6020	81	1362	10	fashion
1	2016-05-23 15:56:20	6043	6029	71	38	19	movies
2	2016-05-23 15:58:11	6038	5721	23	766	19	movies
3	2016-05-23 16:00:45	6046	5877	60	999	18	movies
4	2016-05-23 16:04:28	6037	5891	111	1674	1	automotive

Next, We are going to add an extra column to the ad-clicks table and make it equal to 1. We do so to record the fact that each ROW is 1 ad-click. You will see how this will become useful when we sum up this column to find how many ads did a user click.

```
In [5]:adclicksDF['adCount'] = 1
```

Let us display the first 5 lines of adclicksDF and see if a new column has been added:

In [6]:
```
adclicksDF.head(n=5)
```

Out[6]:

	timestamp	txID	userSessionid	teamid	userid	adID	adCategory	adCount
0	2016-05-23 15:52:24	6045	6020	81	1362	10	fashion	1
1	2016-05-23 15:56:20	6043	6029	71	38	19	movies	1
2	2016-05-23 15:58:11	6038	5721	23	766	19	movies	1
3	2016-05-23 16:00:45	6046	5877	60	999	18	movies	1
4	2016-05-23 16:04:28	6037	5891	111	1674	1	automotive	1

Next, let us read the contents of the file buy-clicks.csv. As before, the following commands read in the CSV file in a table format and removes any extra whitespaces. So, if the CSV contained ' userid ' it becomes 'userid'.

Note: that you must change the path to buy-clicks.csv to the location on your machine, if you want to run this command on your machine.

In [7]:
```
buyclicksDF = pd.read_csv('/Users/aloksingh/Downloads/capstone/second/courseraDataSimulation/buy-clicks.csv')
buyclicksDF = buyclicksDF.rename(columns=lambda x: x.strip()) #removes whitespaces from headers
```

Let us display the first 5 lines of buyclicksDF:

In [8]:
```
buyclicksDF.head(n=5)
```

Out[8]:

	timestamp	txID	userSessionid	team	userid	buyid	price
0	2016-05-23 16:19:14	6086	5692	115	2472	3	4.99
1	2016-05-23 16:19:14	6087	5690	33	2391	1	1.99
2	2016-05-23 16:49:14	6120	5693	6	2156	4	9.99
3	2016-05-23 16:49:14	6121	5689	69	502	0	0.99
4	2016-05-23 16:49:14	6122	5691	97	2198	2	2.99

Feature Selection

For this exercise, we can choose from buyclicksDF, the 'price' of each app that a user purchases as an attribute that captures user's purchasing behavior. The following command selects 'userid' and 'price' and drops all other columns that we do not want to use at this stage.

In [9]:
```
userPurchases = buyclicksDF[['userid','price']] #select only userid and price
userPurchases.head(n=5)
```

Out[9]:

	userid	price
0	2472	4.99

1	2391	1.99
2	2156	9.99
3	502	0.99
4	2198	2.99

Similarly, from the adclicksDF, we will use the 'adCount' as an attribute that captures user's inclination to click on ads. The following command selects 'userid' and 'adCount' and drops all other columns that we do not want to use at this stage.

```
In [10]: useradClicks = adclicksDF[['userid','adCount']]
```

```
In [11]: useradClicks.head(n=5) #as we saw before, this line displays first five lines
```

Out[11]:

	userid	**adCount**
0	1362	1
1	38	1
2	766	1
3	999	1
4	1674	1

Create the first aggregate feature for clustering

From each of these single ad-clicks per row, we can now generate total ad clicks per user. Let's pick a user with userid = 3. To find out how many ads this user has clicked overall, we have to find each row that contains userid = 3, and report the total number of such rows. The following commands sum the total number of ads per user and rename the columns to be called 'userid' and 'totalAdClicks'.

Note that you may not need to aggregate (e.g. sum over many rows) if you choose a different feature or if your data set already provides the necessary information you need. However, most of the times, we do have to process the raw data before it is ready for training a clustering model. In the end, we want to get one row per user, if we are performing clustering over users.

```
In [12]: adsPerUser = useradClicks.groupby('userid').sum()
adsPerUser = adsPerUser.reset_index()
adsPerUser.columns = ['userid', 'totalAdClicks'] #rename the columns
```

Let us display the first 5 lines of 'adsPerUser' to see if there is a column named 'totalAdClicks' containing total adclicks per user.

In [13]:
```
adsPerUser.head(n=5)
```

Out[13]:

	userid	totalAdClicks
0	3	33
1	4	44
2	9	43
3	17	32
4	20	40

Create the second aggregate feature for clustering

Similar to what we did for adclicks, here we find out how much money in total did each user spend on buying in-app purchases. As an example, let's pick a user with userid = 9. To find out the total money spent by this user, we have to find each row that contains userid = 9, and report the sum of the column'price' of each product they purchased. The following commands sum the total money spent by each user and rename the columns to be called 'userid' and 'revenue'.

Note: that you can also use other aggregates, such as sum of money spent on a specific ad category by a user or on a set of ad categories by each user, game clicks per hour by each user etc. You are free to use any mathematical operations on the fields provided in the CSV files when creating features.

In [14]:
```
revenuePerUser = userPurchases.groupby('userid').sum()
revenuePerUser = revenuePerUser.reset_index()
```

```
revenuePerUser.columns = ['userid', 'revenue'] #rename the columns
```

In [15]:
```
revenuePerUser.head(n=5)
```

Out[15]:

	userid	revenue
0	3	11.97
1	4	27.95
2	9	10.94
3	17	5.97
4	20	69.92

Merge the two tables

Lets see what we have so far. We have a table called revenuePerUser, where each row contains total money a user (with that 'userid') has spent. We also have another table called adsPerUser where each row contains total number of ads a user has clicked. We will use revenuePerUser and adsPerUser as features / attributes to capture our users' behavior.

Let us combine these two attributes (features) so that each row contains both attributes per user. Let's merge these two tables to get one single table we can use for K-Means clustering.

In [16]:
```
combinedDF = adsPerUser.merge(revenuePerUser, on='userid') #userid, adCount, price
```

Let us display the first 5 lines of the merged table. **Note: Depending on what attributes you choose, you may not need to merge tables. You may get all your attributes from a single table.**

In [17]:
```
combinedDF.head(n=5) #display how the merged table looks
```

Out[17]:

	userid	totalAdClicks	revenue
0	3	33	11.97
1	4	44	27.95
2	9	43	10.94
3	17	32	5.97
4	20	40	69.92

Create the final training dataset

Our training data set is almost ready. At this stage we can remove the 'userid' from each row, since 'userid' is a computer generated random number assigned to each user. It does not capture any behavioral aspect of a user. One way to drop the 'userid', is to select the other two columns.

In [18]:
```
trainingDF = combinedDF[['totalAdClicks','revenue']]
trainingDF.head(n=5)
```

Out[18]:

	totalAdClicks	revenue
0	33	11.97
1	44	27.95
2	43	10.94
3	32	5.97
4	40	69.92

Display the dimensions of the training dataset

Display the dimension of the training data set. To display the dimensions of the trainingDF, simply add .shape as a suffix and hit enter.

```
In [19]: trainingDF.shape
Out[19]:
(604, 2)
```

The following two commands convert the tables we created into a format that can be understood by the KMeans.train function.

line[0] refers to the first column. line[1] refers to the second column. **Note:** If you have more than 2 columns in your training table, modify this command by adding line[2], line[3], line[4] ...

```
In [20]: pDF = sqlContext.createDataFrame(trainingDF)
parsedData = pDF.rdd.map(lambda line: array([line[0], line[1]])) #totalAdClicks, revenue
```

Train KMeans model

Here we are creating two clusters as denoted in the second argument.

```
In [21]: my_kmmodel = KMeans.train(parsedData, 2, maxIterations=10, runs=10,
initializationMode="random")
```

```
/usr/local/Cellar/apache-spark/1.6.0/libexec/python/pyspark/mllib/clustering.py:176: UserWarning:
Support for runs is deprecated in 1.6.0. This param will have no effect in 1.7.0.
  "Support for runs is deprecated in 1.6.0. This param will have no effect in 1.7.0.")
```

Display the centers of two clusters formed

```
In [22]: print(my_kmmodel.centers)
```

Out [22]:

```
[array([ 41.29090909,  61.92224242]), array([ 25.15489749,  15.48749431])]
```

Analyze the cluster centers

Each array denotes the center for a cluster:

One Cluster is centered at ... array([25.15489749, 15.48749431])

Other Cluster is centered at ... array([41.29090909, 61.92224242])

First number (field1) in each array refers to mean number of ad-clicks and the second number (field2) is the mean revenue per user for users in that cluster. Compare the 1st number of each

cluster to see how differently users in each cluster behave when it comes to clicking ads. Compare the 2nd number of each cluster to see how differently users in each cluster behave when it comes to buying stuff.

In one cluster, in general, players click on ads much more often (~1.6 times) and spend more money (~4 times) on in-app purchases. Assuming that Eglence Inc. gets paid for showing ads and for hosting in-app purchase items, we can use this information to increase game's revenue by increasing the prices for ads we show to the frequent-clickers, and charge higher fees for hosting the in-app purchase items shown to the higher revenue generating buyers.

Note: This analysis requires you to compare the cluster centers and find any ‘significant’ differences in the corresponding feature values of the centers. The answer to this question will depend on the features you have chosen. Some features help distinguish the clusters remarkably while others may not tell you much. At this point, if you don’t find clear distinguishing patterns, perhaps re-running the clustering model with different numbers of clusters and revising the features you picked would be a good idea.