Chapter 9: K-means Clustering

Ex 2: Clustering Consulting Project - Hack_data - Solutions

A large technology firm needs your help, they've been hacked! Luckily their forensic engineers have grabbed valuable data about the hacks, including information like session time,locations, wpm typing speed, etc. The forensic engineer relates to you what she has been able to figure out so far, she has been able to grab meta data of each session that the hackers used to connect to their servers. These are the features of the data:

- 'Session_Connection_Time': How long the session lasted in minutes
- 'Bytes Transferred': Number of MB transferred during session
- 'Kali_Trace_Used': Indicates if the hacker was using Kali Linux
- 'Servers_Corrupted': Number of server corrupted during the attack
- 'Pages_Corrupted': Number of pages illegally accessed
- 'Location': Location attack came from (Probably useless because the hackers used VPNs)
- 'WPM_Typing_Speed': Their estimated typing speed based on session logs.

The technology firm has 3 potential hackers that perpetrated the attack. Their certain of the first two hackers but they aren't very sure if the third hacker was involved or not. They have requested your help! Can you help figure out whether or not the third suspect had anything to do with the attacks, or was it just two hackers? It's probably not possible to know for sure, but maybe what you've just learned about Clustering can help!

One last key fact, the forensic engineer knows that the hackers trade off attacks. Meaning they should each have roughly the same amount of attacks. For example if there were 100 total attacks, then in a 2 hacker situation each should have about 50 hacks, in a three hacker situation each would have about 33 hacks. The engineer believes this is the key element to solving this, but doesn't know how to distinguish this unlabeled data into groups of hackers.

In [1]:

```
import findspark
findspark.init()
```

In [2]:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('hack_find').getOrCreate()
```

In [3]:

```
from pyspark.ml.clustering import KMeans

# Loads data.
dataset = spark.read.csv("hack_data.csv", header=True,
                         inferSchema=True)
```

In [4]:
```
dataset.head()
```

Out[4]:
```
Row(Session_Connection_Time=8.0, Bytes Transferred=391.09, Kali_Trace_Used=1, S
ervers_Corrupted=2.96, Pages_Corrupted=7.0, Location='Slovenia', WPM_Typing_Spe
ed=72.37)
```

In [5]:
```
dataset.describe().show()
```

```
+-------+-----------------------+------------------+------------------+--------
---------+------------------+-----------+------------------+
|summary|Session_Connection_Time| Bytes Transferred|   Kali_Trace_Used|Servers_
Corrupted|   Pages_Corrupted|   Location|  WPM_Typing_Speed|
+-------+-----------------------+------------------+------------------+--------
---------+------------------+-----------+------------------+
|  count|                    334|               334|               334|
334|               334|        334|               334|
|   mean|     30.008982035928145| 607.2452694610777|0.5119760479041916|5.258502
994011977|10.838323353293413|       null|57.342395209580864|
| stddev|     14.088200614636158|286.33593163576757|0.5006065264451406| 2.30190
693339697|  3.06352633036022|       null| 13.41106336843464|
|    min|                    1.0|              10.0|                 0|
1.0|               6.0|Afghanistan|              40.0|
|    max|                   60.0|            1330.5|                 1|
10.0|              15.0|   Zimbabwe|              75.0|
+-------+-----------------------+------------------+------------------+--------
---------+------------------+-----------+------------------+
```

In [6]:
```
dataset.columns
```

Out[6]:
```
['Session_Connection_Time',
 'Bytes Transferred',
 'Kali_Trace_Used',
 'Servers_Corrupted',
 'Pages_Corrupted',
 'Location',
 'WPM_Typing_Speed']
```

In [7]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [8]:
```
feat_cols = ['Session_Connection_Time',
             'Bytes Transferred',
             'Kali_Trace_Used',
             'Servers_Corrupted',
             'Pages_Corrupted','WPM_Typing_Speed']
```

In [9]:
```
vec_assembler = VectorAssembler(inputCols = feat_cols,
                                outputCol='features')
```

In [10]:
```
final_data = vec_assembler.transform(dataset)
```

In [11]:
```
from pyspark.ml.feature import StandardScaler
```

In [12]:

```
scaler = StandardScaler(inputCol="features",
                        outputCol="scaledFeatures",
                        withStd=True,
                        withMean=False)
```

In [13]:

```
# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(final_data)
```

In [14]:

```
# Normalize each feature to have unit standard deviation.
cluster_final_data = scalerModel.transform(final_data)
```

** Time to find out whether its 2 or 3! **

In [15]:

```
kmeans3 = KMeans(featuresCol='scaledFeatures',k=3)
kmeans2 = KMeans(featuresCol='scaledFeatures',k=2)
```

In [16]:

```
model_k3 = kmeans3.fit(cluster_final_data)
model_k2 = kmeans2.fit(cluster_final_data)
```

In [17]:

```
wssse_k3 = model_k3.computeCost(cluster_final_data)
wssse_k2 = model_k2.computeCost(cluster_final_data)
```

In [18]:

```
print("With K=3")
print("Within Set Sum of Squared Errors = " + str(wssse_k3))
print('--'*30)
print("With K=2")
print("Within Set Sum of Squared Errors = " + str(wssse_k2))
```

```
With K=3
Within Set Sum of Squared Errors = 434.1492898715845
------------------------------------------------------------
With K=2
Within Set Sum of Squared Errors = 601.7707512676716
```

Not much to be gained from the WSSSE, after all, we would expect that as K increases, the WSSSE decreases. We could however continue the analysis by seeing the drop from K=3 to K=4 to check if the clustering favors even or odd numbers. This won't be substantial, but its worth a look:

In [19]:

```
for k in range(2,9):
    kmeans = KMeans(featuresCol='scaledFeatures',k=k)
    model = kmeans.fit(cluster_final_data)
    wssse = model.computeCost(cluster_final_data)
    print("With K={}".format(k))
    print("Within Set Sum of Squared Errors = " + str(wssse))
    print('--'*30)
```

```
With K=2
Within Set Sum of Squared Errors = 601.7707512676716
------------------------------------------------------------
With K=3
Within Set Sum of Squared Errors = 434.1492898715845
------------------------------------------------------------
With K=4
Within Set Sum of Squared Errors = 414.8171173373783
------------------------------------------------------------
With K=5
Within Set Sum of Squared Errors = 247.80143915458297
------------------------------------------------------------
With K=6
Within Set Sum of Squared Errors = 232.65169349742308
------------------------------------------------------------
With K=7
Within Set Sum of Squared Errors = 219.57683951525962
------------------------------------------------------------
With K=8
Within Set Sum of Squared Errors = 206.34236284345502
------------------------------------------------------------
```

** Nothing definitive can be said with the above, but wait! The last key fact that the engineer mentioned was that the attacks should be evenly numbered between the hackers! Let's check with the transform and prediction columns that result form this! Congratulations if you made this connection, it was quite tricky given what we've covered!**

In [20]:

```
model_k3.transform(cluster_final_data).groupBy('prediction').count().show()
```

```
+----------+-----+
|prediction|count|
+----------+-----+
|         1|  167|
|         2|   83|
|         0|   84|
+----------+-----+
```

In [21]:

```
model_k2.transform(cluster_final_data).groupBy('prediction').count().show()
```

```
+----------+-----+
|prediction|count|
+----------+-----+
|         1|  167|
|         0|  167|
+----------+-----+
```

It was 2 hackers, in fact, our clustering algorithm created two equally sized clusters with K=2, no way that is a coincidence!

In [ ]: