CDR anomaly without and with feature normalization

Collapse

Running

By cdsw (http://10.0.0.242.nip.io/cdsw) — Python 2 Session (Base Image v5) — 1 hour ago for running

```
from __future__ import print_function
!echo $PYTHON_PATH

import os, sys
```

import path

```
from pyspark.sql import *
```

create spark sql session

```
myspark = SparkSession\
    .builder\
    .config("spark.executor.instances", 4 ) \
    .config("spark.executor.memory", "10g") \
    .config("spark.executor.cores", 2) \
    .config("spark.dynamicAllocation.maxExecutors", 10) \
    .config("spark.scheduler.listenerbus.eventqueue.size", 10000) \
    .config("spark.sql.parquet.compression.codec", "snappy") \
    .appName("telco_kmeans") \
    .getOrCreate()
```

```
19/01/24 14:27:59 ERROR spark.SparkContext: Error initializing SparkContex
t.
java.lang.IllegalArgumentException: Required executor memory (10240+1024 M
B) is above the max threshold (10240 MB) of this cluster! Please check the
values of 'yarn.scheduler.maximum-allocation-mb' and/or 'yarn.nodemanager.
resource.memory-mb'.
        at org.apache.spark.deploy.yarn.Client.verifyClusterResources(Clie
nt.scala:358)
        at org.apache.spark.deploy.yarn.Client.submitApplication(Client.sc
ala:170)
        at org.apache.spark.scheduler.cluster.YarnClientSchedulerBackend.s
tart(YarnClientSchedulerBackend.scala:57)
        at org.apache.spark.scheduler.TaskSchedulerImpl.start(TaskSchedule
rImpl.scala:164)
        at org.apache.spark.SparkContext.<init>(SparkContext.scala:500)
        at org.apache.spark.api.java.JavaSparkContext.<init>(JavaSparkCont
ext.scala:58)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native M
ethod)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeCon
structorAccessorImpl.java:62)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Deleg
atingConstructorAccessorImpl.java:45)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:247)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:3
57)
        at py4j.Gateway.invoke(Gateway.java:238)
        at py4j.commands.ConstructorCommand.invokeConstructor(ConstructorC
ommand.java:80)
        at py4j.commands.ConstructorCommand.execute(ConstructorCommand.jav
a:69)
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
        at java.lang.Thread.run(Thread.java:745)
```

```
Py4JJavaError: An error occurred while calling None.org.apache.spark.api.ja
: java.lang.IllegalArgumentException: Required executor memory (10240+1024
        at org.apache.spark.deploy.yarn.Client.verifyClusterResources(Clien
        at org.apache.spark.deploy.yarn.Client.submitApplication(Client.sca
        at org.apache.spark.scheduler.cluster.YarnClientSchedulerBackend.st
        at org.apache.spark.scheduler.TaskSchedulerImpl.start(TaskScheduler
        at org.apache.spark.SparkContext.<init>(SparkContext.scala:500)
        at org.apache.spark.api.java.JavaSparkContext.<init>(JavaSparkConte
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Me
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeCons
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Delega
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:247)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:35
        at py4j.Gateway.invoke(Gateway.java:238)
        at py4j.commands.ConstructorCommand.invokeConstructor(ConstructorCo
        at py4j.commands.ConstructorCommand.execute(ConstructorCommand.java
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
        at java.lang.Thread.run(Thread.java:745)
```

```
Py4JJavaErrorTraceback (most recent call last)
in engine
----> 1 myspark = SparkSession    .builder    .config("spark.executor.insta

/opt/cloudera/parcels/SPARK2/lib/spark2/python/pyspark/sql/session.py in ge
    171                     for key, value in self._options.items():
    172                         sparkConf.set(key, value)
--> 173                     sc = SparkContext.getOrCreate(sparkConf)
    174                     # This SparkContext may be an existing one.
    175                     for key, value in self._options.items():

/opt/cloudera/parcels/SPARK2/lib/spark2/python/pyspark/context.py in getOrC
    341         with SparkContext._lock:
    342             if SparkContext._active_spark_context is None:
--> 343                 SparkContext(conf=conf or SparkConf())
    344             return SparkContext._active_spark_context
    345

/opt/cloudera/parcels/SPARK2/lib/spark2/python/pyspark/context.py in __init
    116         try:
    117             self._do_init(master, appName, sparkHome, pyFiles, envi
--> 118                           conf, jsc, profiler_cls)
    119         except:
    120             # If an error occurs, clean up in order to allow future
```

```
/opt/cloudera/parcels/SPARK2/lib/spark2/python/pyspark/context.py in _do_in
    178
    179         # Create the Java SparkContext through Py4J
--> 180         self._jsc = jsc or self._initialize_context(self._conf._jco
    181         # Reset the SparkConf to the one actually used by the Spark
    182         self._conf = SparkConf(_jconf=self._jsc.sc().conf())

/opt/cloudera/parcels/SPARK2/lib/spark2/python/pyspark/context.py in _initi
    280         Initialize SparkContext in function to allow subclass speci
    281         """
--> 282         return self._jvm.JavaSparkContext(jconf)
    283
    284     @classmethod

/usr/local/lib/python2.7/site-packages/py4j/java_gateway.pyc in __call__(se
   1523         answer = self._gateway_client.send_command(command)
   1524         return_value = get_return_value(
-> 1525             answer, self._gateway_client, None, self._fqn)
   1526
   1527         for temp_arg in temp_args:

/usr/local/lib/python2.7/site-packages/py4j/protocol.pyc in get_return_valu
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling None.org.apache.spark.api.ja
: java.lang.IllegalArgumentException: Required executor memory (10240+1024
        at org.apache.spark.deploy.yarn.Client.verifyClusterResources(Clien
        at org.apache.spark.deploy.yarn.Client.submitApplication(Client.sca
        at org.apache.spark.scheduler.cluster.YarnClientSchedulerBackend.st
        at org.apache.spark.scheduler.TaskSchedulerImpl.start(TaskScheduler
        at org.apache.spark.SparkContext.<init>(SparkContext.scala:500)
        at org.apache.spark.api.java.JavaSparkContext.<init>(JavaSparkConte
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Me
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeCons
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Delega
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:247)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:35
        at py4j.Gateway.invoke(Gateway.java:238)
        at py4j.commands.ConstructorCommand.invokeConstructor(ConstructorCo
        at py4j.commands.ConstructorCommand.execute(ConstructorCommand.java
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
        at java.lang.Thread.run(Thread.java:745)
```

```
from __future__ import print_function
!echo $PYTHON_PATH

import os, sys
```

import path

```
from pyspark.sql import *
```

create spark sql session

```
myspark = SparkSession\
    .builder\
    .config("spark.executor.instances", 4 ) \
    .config("spark.executor.memory", "8g") \
    .config("spark.executor.cores", 2) \
    .config("spark.dynamicAllocation.maxExecutors", 10) \
    .config("spark.scheduler.listenerbus.eventqueue.size", 10000) \
    .config("spark.sql.parquet.compression.codec", "snappy") \
    .appName("telco_kmeans") \
    .getOrCreate()
sc = myspark.sparkContext
import time
print ( time.time())
```

```
1548340122.65
```

```
sc.setLogLevel("ERROR")
print ( myspark )
```

```
<pyspark.sql.session.SparkSession object at 0x7f58252c2650>
```

make spark print text instead of octal

```
myspark.sql("SET spark.sql.parquet.binaryAsString=true")
```

```
DataFrame[key: string, value: string]
```

read in the data file from HDFS

```
dfpfc = myspark.read.parquet ( "/user/hive/warehouse/cdranomaly_p")
```

print number of rows and type of object

```
print ( dfpfc.count() )
```

```
49999
```

```
print  ( dfpfc )
```

```
DataFrame[billidnum: int, sourcenm: string, destnm: string, duration: int,
mytimestamp: int, terminationcode: int]
```

create a table name to use for queries

```
dfpfc.createOrReplaceTempView("cdrs")
myspark.sql ("refresh table cdrs")
```

```
DataFrame[]
```

run a query

```
fcout=myspark.sql('select avg(duration ) from cdrs')
fcout.show(5)
```

```
+-----------------+
|    avg(duration)|
+-----------------+
|969.6539130782616|
+-----------------+
```

create a dataframe with valid rows

```
mydf=myspark.sql('select billidnum as label, sourcenm, duration, mytimestam
```

```
mydf.show(5)
```

```
+-----+----------+--------+-----------+---------------+
|label|  sourcenm|duration|mytimestamp|terminationcode|
+-----+----------+--------+-----------+---------------+
|    1|8885551418|    3106|      65802|              8|
|    2|8885550630|     527|      13734|              2|
|    3|8885551297|     520|       7053|              3|
|    4|8885552271|     357|      74181|              8|
|    5|8885552651|     155|      61500|              2|
+-----+----------+--------+-----------+---------------+
only showing top 5 rows
```

need to convert from text field to numeric this is a common requirement when using sparkML from pyspark.ml.feature import StringIndexer

this will convert each unique string into a numeric indexer = StringIndexer(inputCol=”sourcenm”, outputCol=”sourcenumint”) indexed = indexer.fit(mydf).transform(mydf) indexed.show(5) now we need to create a “label” and “features” input for using the sparkML library

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
assembler = VectorAssembler(
    inputCols=[ "duration", "mytimestamp", "terminationcode"],
    outputCol="features")
output = assembler.transform(mydf)
```

note the column headers - label and features are keywords

```
print ( output.show(3) )
```

```
+-----+----------+--------+-----------+---------------+-------------------
-+
|label|  sourcenm|duration|mytimestamp|terminationcode|            feature
s|
+-----+----------+--------+-----------+---------------+-------------------
-+
|    1|8885551418|    3106|      65802|              8|[3106.0,65802.0,8.
0]|
|    2|8885550630|     527|      13734|              2| [527.0,13734.0,2.
0]|
|    3|8885551297|     520|       7053|              3|  [520.0,7053.0,3.
0]|
+-----+----------+--------+-----------+---------------+-------------------
-+
only showing top 3 rows
```

```
None
```

use the kmeans clustering - do not write it yourself :-)

```
from pyspark.ml.clustering import KMeans
```

try 10 different centers

we will start with 10 cluster centers run the model and then come back here, change the 10 to 15 highlight from this line to the bottom and select “run selected lines” it will then see the cluster in the upper right hand corner of the scatter plot

```
kmeans = KMeans().setK(20).setSeed(1)
```

run the model

```
model = kmeans.fit(output)
```

Evaluate clustering by computing Within Set Sum of Squared Errors.

```
wssse = model.computeCost(output)
print("Within Set Sum of Squared Errors = " + str(wssse))
```

```
Within Set Sum of Squared Errors = 1.60491991991e+11
```

Shows the result.

```
centers = model.clusterCenters()
print("Cluster Centers: ")
```

```
Cluster Centers:
```

we know duration in hundreds, timestamp in thousands and term code 1 to 10

```
for center in centers:
    print(center)
```

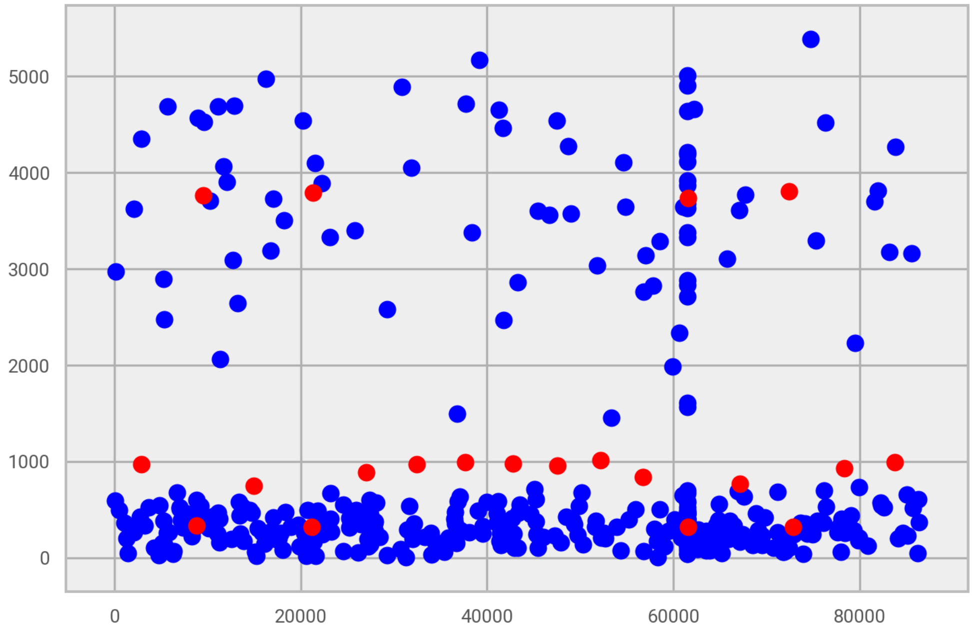
```
[  9.70035515e+02   3.24430786e+04   3.28611333e+00]
[  9.26860637e+02   7.83832807e+04   3.26086957e+00]
[  966.46674017  2897.41087835     3.24402793]
[  8.38402229e+02   5.67222536e+04   3.27264282e+00]
[  1.00930078e+03   5.22109438e+04   3.33671119e+00]
[  3.15662269e+02   2.11826953e+04   2.98724714e+00]
[  7.64876599e+02   6.71528956e+04   3.03342963e+00]
[  3.78973748e+03   2.13126525e+04   4.54172989e+00]
[  7.44477371e+02   1.49948175e+04   3.18750000e+00]
[  3.29454698e+02   8.80180789e+03   2.95134228e+00]
[  3.72994976e+03   6.15776114e+04   4.47874624e+00]
[  3.18504284e+02   6.15559473e+04   3.00010711e+00]
[  3.80401958e+03   7.24016401e+04   4.55421687e+00]
[  9.93503205e+02   8.37587696e+04   3.23918269e+00]
[  8.86965953e+02   2.70560084e+04   3.19166029e+00]
[  9.92982881e+02   3.76848685e+04   3.31398747e+00]
[  9.55397722e+02   4.75879238e+04   3.33552343e+00]
[  9.76164313e+02   4.27869867e+04   3.27385537e+00]
[  3.20301178e+02   7.28596703e+04   3.01132246e+00]
[  3.75790525e+03   9.55003353e+03   4.56997085e+00]
```

now create pretty graph %matplotlib inline

```
import matplotlib.pyplot as plt
```

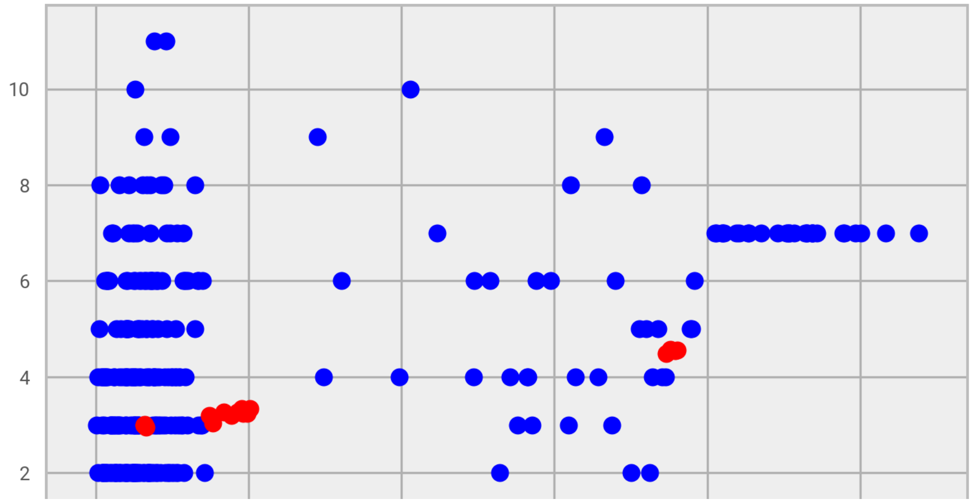
== y= duration vs x= timestamp print (output.take(3))

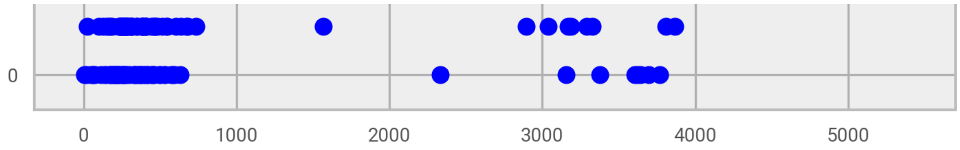
```
def plotit(numpts):
 for row in output.take(numpts):
    plt.scatter(row[3],row[2], color=['blue'])
 for center in centers:
    plt.scatter(center[1],center[0],color=['red'])
 plt.show()
plotit(400)
```



y= termination code vs x=duration print (output.take(10))

```
def plotit(numpts):
 for row in output.take(numpts):
    plt.scatter(row[2],row[4], color=['blue'])
 for center in centers:
  plt.scatter(center[0],center[2],color=['red'])
 plt.show()
plotit(400)
```





compute distance from each point to the center it was assigned the anomalies are the points that are the greatest distance from the assigned cluster center

score the data

```
df_pred = model.transform(output)
df_pred.show(10)
```

```
+-----+----------+--------+-----------+---------------+-------------------
-+----------+
|label|  sourcenm|duration|mytimestamp|terminationcode|            feature
s|prediction|
+-----+----------+--------+-----------+---------------+-------------------
-+----------+
|    1|8885551418|    3106|      65802|              8|[3106.0,65802.0,8.
0]|         6|
|    2|8885550630|     527|      13734|              2| [527.0,13734.0,2.
0]|         8|
|    3|8885551297|     520|       7053|              3|  [520.0,7053.0,3.
0]|         9|
|    4|8885552271|     357|      74181|              8| [357.0,74181.0,8.
0]|        18|
|    5|8885552651|     155|      61500|              2| [155.0,61500.0,2.
0]|        11|
|    6|8885550860|     279|      42494|              3| [279.0,42494.0,3.
0]|        17|
|    7|8885550239|     456|      41095|              2| [456.0,41095.0,2.
0]|        17|
|    8|8885551395|     255|      17840|              4| [255.0,17840.0,4.
0]|         8|
|    9|8885550440|    1987|      59904|              4|[1987.0,59904.0,4.
0]|        11|
|   10|8885551164|    3378|      38405|              3|[3378.0,38405.0,3.
0]|        15|
+-----+----------+--------+-----------+---------------+-------------------
-+----------+
only showing top 10 rows
```

distance is sqrt ((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2)

create a dataframe with valid rows

```
mydf=myspark.sql('select billidnum as label, sourcenm, duration*10 dur, myt
```

mydf=myspark.sql(‘select billidnum as label, sourcenm, duration, mytimestamp, terminationcode from cdrs’)

```
mydf.show(5)
```

```
+-----+----------+-----+------+---+
|label|  sourcenm|  dur|    ts| tc|
+-----+----------+-----+------+---+
|    1|8885551418|31060|658.02| 80|
|    2|8885550630| 5270|137.34| 20|
|    3|8885551297| 5200| 70.53| 30|
|    4|8885552271| 3570|741.81| 80|
|    5|8885552651| 1550| 615.0| 20|
+-----+----------+-----+------+---+
only showing top 5 rows
```

need to convert from text field to numeric this is a common requirement when using sparkML from pyspark.ml.feature import StringIndexer

this will convert each unique string into a numeric indexer = StringIndexer(inputCol=”sourcenm”, outputCol=”sourcenumint”) indexed = indexer.fit(mydf).transform(mydf) indexed.show(5) now we need to create a “label” and “features” input for using the sparkML library

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
assembler = VectorAssembler(
    #inputCols=[ "duration", "mytimestamp", "terminationcode"],
    inputCols=[ "dur", "ts", "tc"],
    outputCol="features")
output = assembler.transform(mydf)
```

note the column headers - label and features are keywords

```
print ( output.show(3) )
```

```
+-----+----------+-----+------+---+--------------------+
|label|  sourcenm|  dur|    ts| tc|            features|
+-----+----------+-----+------+---+--------------------+
|    1|8885551418|31060|658.02| 80|[31060.0,658.02,8...|
|    2|8885550630| 5270|137.34| 20|[5270.0,137.34,20.0]|
|    3|8885551297| 5200| 70.53| 30| [5200.0,70.53,30.0]|
+-----+----------+-----+------+---+--------------------+
only showing top 3 rows

None
```

use the kmeans clustering - do not write it yourself :-)

```
from pyspark.ml.clustering import KMeans
```

try 10 different centers

we will start with 10 cluster centers run the model and then come back here, change the 10 to 15 highlight from this line to the bottom and select “run selected lines” it will then see the cluster in the upper right hand corner of the scatter plot

```
kmeans = KMeans().setK(20).setSeed(1)
```

run the model

```
model = kmeans.fit(output)
```

Evaluate clustering by computing Within Set Sum of Squared Errors.

```
wssse = model.computeCost(output)
print("Within Set Sum of Squared Errors = " + str(wssse))
```

```
Within Set Sum of Squared Errors = 18849946968.1
```

Shows the result.

```
centers = model.clusterCenters()
print("Cluster Centers: ")
```

```
Cluster Centers:
```

we know duration in hundreds, timestamp in thousands and term code 1 to 10

```
for center in centers:
    print(center)
```

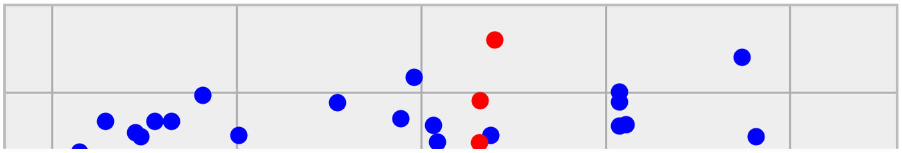
```
[ 506.14206721  466.4350638    29.17907274]
[ 7280.28340081   475.82004049    30.12145749]
[  3.25092672e+04   4.61036475e+02   3.07699443e+01]
[ 24169.08026756    463.22172241     30.06688963]
[ 40939.82630273    466.54719603     61.14143921]
[ 3119.0625881    467.04272625    30.26924161]
[ 55776.5497076     479.22505848     70.        ]
[ 49115.78215527    463.36923523     70.        ]
[ 15553.47058824    480.17652941     35.05882353]
[  2.99701283e+04   4.76952310e+02   2.90267380e+01]
[ 1453.1011781    461.21900901    29.86659737]
[ 5795.88370565   466.07461235    31.01182654]
[ 3912.10642317   462.15946946    29.88507557]
[ 4780.01189061   462.59418549    30.2516845 ]
[ 27254.93333333    462.53710667     30.93333333]
[  3.50424011e+04   4.64267653e+02   3.11780576e+01]
[ 2303.96433666   467.96522967    30.021398   ]
[ 44529.7392767     463.26500421     70.        ]
[ 20472.97235023    449.13741935     29.10138249]
[  3.78374828e+04   4.72192426e+02   2.98321892e+01]
```

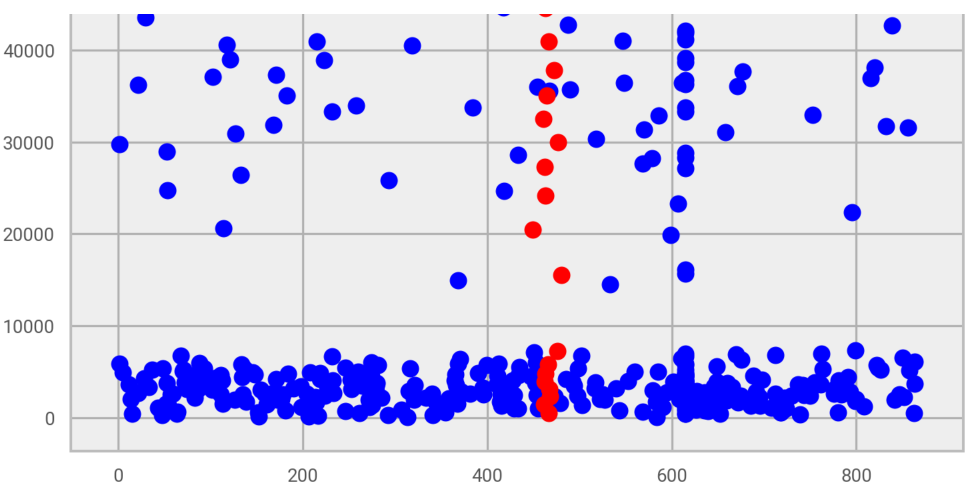
now create pretty graph %matplotlib inline

```
import matplotlib.pyplot as plt
```

=== y= duration vs x= timestamp print (output.take(3))

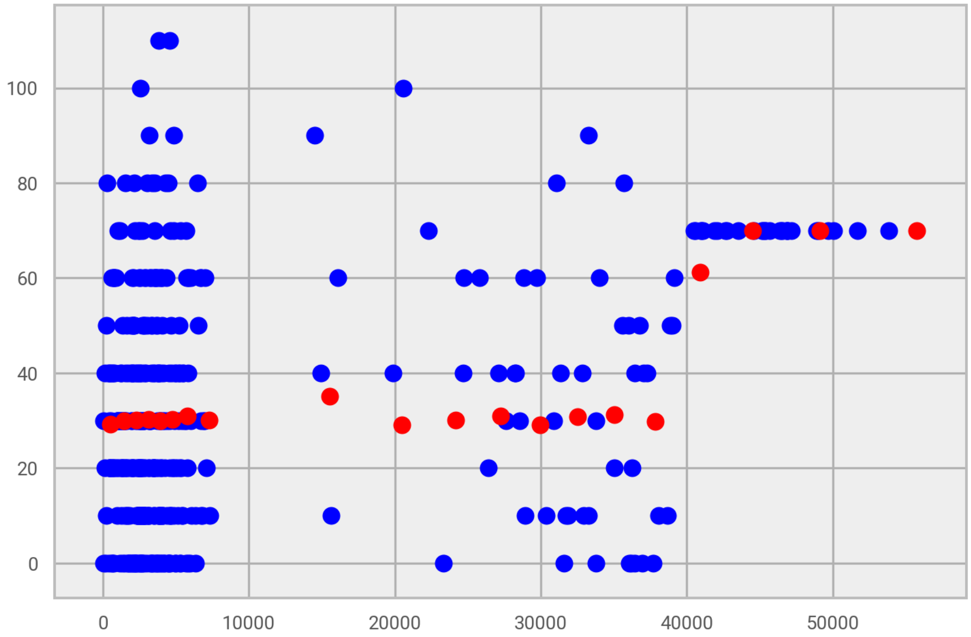
```
def plotit(numpts):
 for row in output.take(numpts):
    plt.scatter(row[3],row[2], color=['blue'])
 for center in centers:
    plt.scatter(center[1],center[0],color=['red'])
 plt.show()
plotit(400)
```





y= termination code vs x=duration print (output.take(10))

```
def plotit(numpts):
 for row in output.take(numpts):
    plt.scatter(row[2],row[4], color=['blue'])
 for center in centers:
  plt.scatter(center[0],center[2],color=['red'])
 plt.show()
plotit(400)
```



compute distance from each point to the center it was assigned the anomalies are the points that are the greatest distance from the assigned cluster center

score the data

```
df_pred = model.transform(output)
df_pred.show(10)
```

```
+-----+----------+-----+------+---+--------------------+----------+
|label|  sourcenm|  dur|    ts| tc|            features|prediction|
+-----+----------+-----+------+---+--------------------+----------+
|    1|8885551418|31060|658.02| 80|[31060.0,658.02,8...|         9|
|    2|8885550630| 5270|137.34| 20|[5270.0,137.34,20.0]|        13|
|    3|8885551297| 5200| 70.53| 30| [5200.0,70.53,30.0]|        13|
|    4|8885552271| 3570|741.81| 80|[3570.0,741.81,80.0]|        12|
|    5|8885552651| 1550| 615.0| 20| [1550.0,615.0,20.0]|        10|
|    6|8885550860| 2790|424.94| 30|[2790.0,424.94,30.0]|         5|
|    7|8885550239| 4560|410.95| 20|[4560.0,410.95,20.0]|        13|
|    8|8885551395| 2550| 178.4| 40| [2550.0,178.4,40.0]|        16|
|    9|8885550440|19870|599.04| 40|[19870.0,599.04,4...|        18|
|   10|8885551164|33780|384.05| 30|[33780.0,384.05,3...|        15|
+-----+----------+-----+------+---+--------------------+----------+
only showing top 10 rows
```

distance is sqrt ((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2)

create a dataframe with valid rows

```
mydf=myspark.sql('select billidnum as label, sourcenm, duration*10 dur, myt
```

mydf=myspark.sql('select billidnum as label, sourcenm, duration, mytimestamp, terminationcode from cdrs')

```
mydf.show(5)
```

```
+-----+----------+-----+------+---+
|label|  sourcenm|  dur|    ts| tc|
+-----+----------+-----+------+---+
|    1|8885551418|31060|658.02|800|
|    2|8885550630| 5270|137.34|200|
|    3|8885551297| 5200| 70.53|300|
|    4|8885552271| 3570|741.81|800|
|    5|8885552651| 1550| 615.0|200|
+-----+----------+-----+------+---+
only showing top 5 rows
```

need to convert from text field to numeric this is a common requirement when using sparkML from pyspark.ml.feature import StringIndexer

this will convert each unique string into a numeric indexer = StringIndexer(inputCol="sourcenm", outputCol="sourcenumint") indexed = indexer.fit(mydf).transform(mydf) indexed.show(5) now we need to create a "label" and "features" input for using the sparkML library

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
assembler = VectorAssembler(
    #inputCols=[ "duration", "mytimestamp", "terminationcode"],
    inputCols=[ "dur", "ts", "tc"],
    outputCol="features")
output = assembler.transform(mydf)
```

note the column headers - label and features are keywords

```
print ( output.show(3) )
```

```
+-----+----------+-----+------+---+--------------------+
|label|  sourcenm|  dur|    ts| tc|            features|
+-----+----------+-----+------+---+--------------------+
|    1|8885551418|31060|658.02|800|[31060.0,658.02,8...|
|    2|8885550630| 5270|137.34|200|[5270.0,137.34,20...|
|    3|8885551297| 5200| 70.53|300|[5200.0,70.53,300.0]|
+-----+----------+-----+------+---+--------------------+
only showing top 3 rows

None
```

use the kmeans clustering - do not write it yourself :-)

```
from pyspark.ml.clustering import KMeans
```

try 10 different centers

we will start with 10 cluster centers run the model and then come back here, change the 10 to 15 highlight from this line to the bottom and select “run selected lines” it will then see the cluster in the upper right hand corner of the scatter plot

```
kmeans = KMeans().setK(20).setSeed(1)
```

run the model

```
model = kmeans.fit(output)
```

Evaluate clustering by computing Within Set Sum of Squared Errors.

```
wssse = model.computeCost(output)
print("Within Set Sum of Squared Errors = " + str(wssse))
```

```
Within Set Sum of Squared Errors = 24418351287.4
```

Shows the result.

```
centers = model.clusterCenters()
print("Cluster Centers: ")
```

```
Cluster Centers:
```

we know duration in hundreds, timestamp in thousands and term code 1 to 10

```
for center in centers:
    print(center)
```

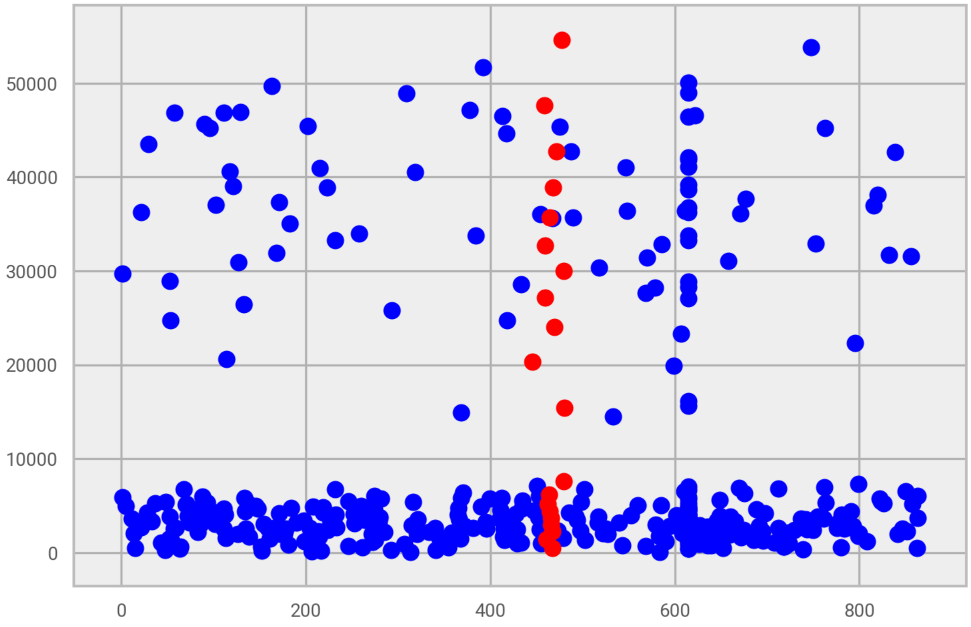
```
[ 493.04680665  467.54765092  289.720035  ]
[ 6145.92679493   463.95210699   312.90473956]
[ 27181.81699346    459.5880915     312.02614379]
[ 38852.86298569    467.69107703    369.87048398]
[ 3668.84990587   465.57063666   302.5329454 ]
[ 54615.15789474    477.70166316    700.        ]
[ 35666.88311688    464.70821238    304.73644003]
[ 42750.89521166    471.85262318    700.        ]
[ 15432.83950617    480.10549383    354.32098765]
[ 20354.21176471    445.26122353    287.76470588]
[ 1415.70707071   460.93564574   299.69336219]
[ 7606.58267717   479.36795276   305.82677165]
[ 2966.44808743   465.67881187   305.54254489]
[ 5169.57027027   462.24010811   306.13513514]
[ 24048.31918506    469.11556876    297.62308998]
[ 47602.8314799    458.4227716    700.        ]
[ 4384.27652066   464.25483081   292.90014532]
[ 29986.3814433     479.40484536    288.35051546]
[ 2222.31252948   467.21554315   298.49080333]
[ 32725.49630845    459.6416735     311.07465135]
```

now create pretty graph %matplotlib inline

```
import matplotlib.pyplot as plt
```

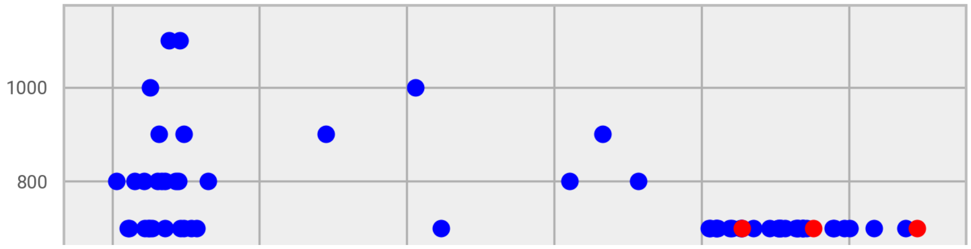
== y= duration vs x= timestamp print (output.take(3))

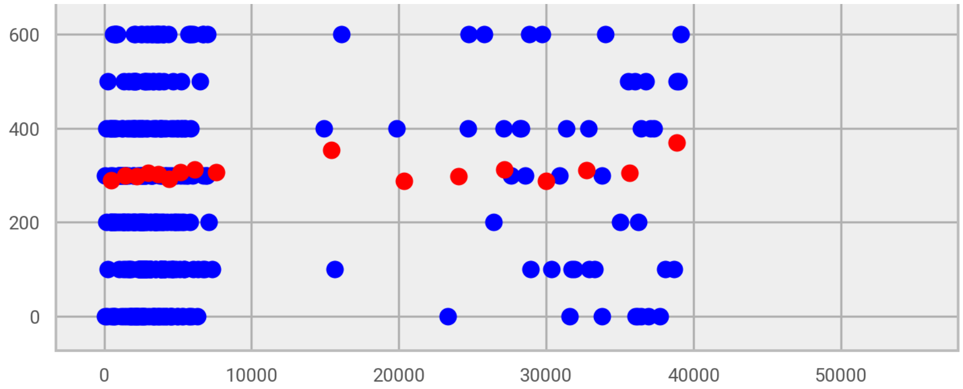
```
def plotit(numpts):
 for row in output.take(numpts):
    plt.scatter(row[3],row[2], color=['blue'])
 for center in centers:
    plt.scatter(center[1],center[0],color=['red'])
 plt.show()
plotit(400)
```



y= termination code vs x=duration print (output.take(10))

```
def plotit(numpts):
 for row in output.take(numpts):
    plt.scatter(row[2],row[4], color=['blue'])
 for center in centers:
  plt.scatter(center[0],center[2],color=['red'])
 plt.show()
plotit(400)
```





compute distance from each point to the center it was assigned the anomalies are the points that are the greatest distance from the assigned cluster center

score the data

```
df_pred = model.transform(output)
df_pred.show(10)
```

```
+-----+----------+-----+------+---+--------------------+----------+
|label|  sourcenm|  dur|    ts| tc|            features|prediction|
+-----+----------+-----+------+---+--------------------+----------+
|    1|8885551418|31060|658.02|800|[31060.0,658.02,8...|        17|
|    2|8885550630| 5270|137.34|200|[5270.0,137.34,20...|        13|
|    3|8885551297| 5200| 70.53|300|[5200.0,70.53,300.0]|        13|
|    4|8885552271| 3570|741.81|800|[3570.0,741.81,80...|         4|
|    5|8885552651| 1550| 615.0|200|[1550.0,615.0,200.0]|        10|
|    6|8885550860| 2790|424.94|300|[2790.0,424.94,30...|        12|
|    7|8885550239| 4560|410.95|200|[4560.0,410.95,20...|        16|
|    8|8885551395| 2550| 178.4|400|[2550.0,178.4,400.0]|        18|
|    9|8885550440|19870|599.04|400|[19870.0,599.04,4...|         9|
|   10|8885551164|33780|384.05|300|[33780.0,384.05,3...|        19|
+-----+----------+-----+------+---+--------------------+----------+
only showing top 10 rows
```

distance is sqrt ((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2)

```
df_pred = model.transform(output)
df_pred.show(10)
```

```
+-----+----------+-----+------+---+--------------------+----------+
|label|  sourcenm|  dur|    ts| tc|            features|prediction|
+-----+----------+-----+------+---+--------------------+----------+
|    1|8885551418|31060|658.02|800|[31060.0,658.02,8...|        17|
|    2|8885550630| 5270|137.34|200|[5270.0,137.34,20...|        13|
|    3|8885551297| 5200| 70.53|300|[5200.0,70.53,300.0]|        13|
|    4|8885552271| 3570|741.81|800|[3570.0,741.81,80...|         4|
|    5|8885552651| 1550| 615.0|200|[1550.0,615.0,200.0]|        10|
|    6|8885550860| 2790|424.94|300|[2790.0,424.94,30...|        12|
|    7|8885550239| 4560|410.95|200|[4560.0,410.95,20...|        16|
|    8|8885551395| 2550| 178.4|400|[2550.0,178.4,400.0]|        18|
|    9|8885550440|19870|599.04|400|[19870.0,599.04,4...|         9|
|   10|8885551164|33780|384.05|300|[33780.0,384.05,3...|        19|
+-----+----------+-----+------+---+--------------------+----------+
only showing top 10 rows
```