Let's Build An Automated Anomalous DB Activity Detector
Using Machine Learning

Demonstration For My East Coast Oracle (ECO) Conference Presentation On November 2, 2020

FREE Machine Learning E-Course For Oracle Professionals https://www.orapub.com/ml-ecourse (https://www.orapub.com/ml-ecourse)

Author: Craig Shallahamer, craig@orapub.com (mailto:craig@orapub.com)
Latest: 30-Oct-2020 v2n

The Process And Objective

- The objective of this project is to build unsupervised single cluster machine learning model to detect an anomalous Oracle performance situation that warrent an analyst's attention... before the phone starts ringing!
- Build a k-means one cluster unsupervised model based on all available AWR dba_hist_sysmetric_summary data
- Determine the anomaly distance threshold in multi-dimensional space
- Check if most recent AWR snap distance exceed the threshold, thereby being an anomoly
- If so create chart and alert

Key Topics

- Multi-dimensional space
- K-means clusters
- Distance to centroid in a multi-dimensional space
- How to determine an anomaly threshold
- Center and scale data
- Dimensional reduction
- Denomalizing normalized data using Python
- Charting and saving output to file
- General Python functions and testing

In [6]:

```
testing = True
print(testing)
```

```
True
```

In [7]:

```
print("Loading libraries", end=" ")

import numpy as np   # To do array and math stuff
import pandas as pd  # To do dataframe and math stuff
import matplotlib    # To do plots
import os            # To access your local OS
import sklearn       # The core ML algorithms
import pickle        # file IO

from datetime import datetime, timedelta
from matplotlib import pyplot as plt
from sklearn import preprocessing
from collections import Counter
from numpy import unique

print("done.")
```

```
Loading libraries done.
```

In [8]:

```
# Core Settings

print("Making core settings")

#pd.set_option('display.max_row', 1000)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', 50)
np.set_printoptions(precision=3)
low_memory=False

baseURL = "http://"  # This should always work

os.chdir("/Users/anathale/Desktop/AIML/AWRAnalysis/testing")       # <

print("   ", baseURL)
print("   ", os.getcwd())
print("\n    IMPORTANT: Make sure these directories exist:")
print("               " + str(os.getcwd()) + "/pypics/kmeans")
print("               " + str(os.getcwd()) + "/pypics/MiniBatchKMeans'

print("\nDone.")
```

```
Making core settings
    http:// (http://)
    /Users/anathale/Desktop/AIML/AWRAnalysis/testing

    IMPORTANT: Make sure these directories exist:
               /Users/anathale/Desktop/AIML/AWRAnalysis/testing/pypic
s/kmeans
               /Users/anathale/Desktop/AIML/AWRAnalysis/testing/pypic
s/MiniBatchKMeans

Done.
```

In [9]:

```
# Load lead data, Read the CSV file into dataframe, leadsDF

def loadData(csvFN_in, verbose_in):

    if verbose_in:
        print("Loading data")
        print("   "+ str(csvFN_in))

    try:
        print("   Loading data from local machine", end="...")
        dataDF = pd.read_csv(csvFN_in)
        print("done.")
    except:
        print("not found.")
        print("   Retreiving from base URL", end="...")
        URLFN  = baseURL + csvFN_in
        dataDF = pd.read_csv(URLFN)
        print("done.")
        print("   Saving file on local macine", end="...")
        dataDF.to_csv (csvFN_in, index=None, header=True)
        print("done.")

    print("   Shape", dataDF.shape)

    return(dataDF)

if testing:
    print("Testing Function: loadData")
    testDF = loadData("dmwperfenv.csv",True)
    print()
    print(testDF.shape)
    print(testDF.head(4))

print("\nDone.")
```

```
Testing Function: loadData
Loading data
   dmwperfenv.csv
   Loading data from local machine...done.
   Shape (88006, 16)

(88006, 16)
   SNAP_ID       DBID  INSTANCE_NUMBER     BEGIN_TIME       END_TIME
INTSIZE  \
0     5074  882962339                1  7/10/20 12:59  7/10/20 13:59
360013
1     5042  882962339                1    7/9/20 4:59    7/9/20 5:59
359937
2     5026  882962339                1   7/8/20 12:59   7/8/20 13:59

360004
3     5006  882962339                1   7/7/20 18:59   7/7/20 19:59
360039

   GROUP_ID  METRIC_ID              METRIC_NAME                   MET
RIC_UNIT  \
0         2       2000  Buffer Cache Hit Ratio  % (LogRead - PhyRea
d)/LogRead
1         2       2000  Buffer Cache Hit Ratio  % (LogRead - PhyRea
d)/LogRead
2         2       2000  Buffer Cache Hit Ratio  % (LogRead - PhyRea
d)/LogRead
3         2       2000  Buffer Cache Hit Ratio  % (LogRead - PhyRea
d)/LogRead
```

```
   NUM_INTERVAL  MINVAL      MAXVAL    AVERAGE  STANDARD_DEVIATION  \
0            60       0  100.000000  99.997313            0.006420   
1            60       0  100.000000  99.810813            0.885308   
2            60       0   99.908707  99.730937            0.100266   
3            60       0  100.000000  99.815087            1.419464   

   SUM_SQUARES  
0  599967.7603  
1  597778.1490  
2  596776.1753  
3  597901.9761  

Done.
```

In [10]:

```
# Cleanup the features

def cleanupFeatures(df_in, features_in, verbose_in):

    if verbose_in:
        print("Cleaning features", df_in.shape)

    for featureName in features_in:
        df_in[featureName] = [x.replace(" ", "") for x in df_in[featu
        df_in[featureName] = [x.replace("'", "") for x in df_in[featu
        #df_in[featureName] = df_in[featureName].str.lower()

    if verbose_in:
        print("done.")

    return(df_in)

if testing:
    print("Testing Function: cleanupFeatures")
    print("\n    Before (METRIC_NAME)")
    print(testDF['METRIC_NAME'].head(4))
    print()
    testDF = cleanupFeatures(testDF.copy(), ['METRIC_NAME'], True)
    print("\n    After (METRIC_NAME)")
    print(testDF['METRIC_NAME'].head(4))

print("\nDone.")
```

```
Testing Function: cleanupFeatures

    Before (METRIC_NAME)
0     Buffer Cache Hit Ratio
1     Buffer Cache Hit Ratio
2     Buffer Cache Hit Ratio
3     Buffer Cache Hit Ratio
Name: METRIC_NAME, dtype: object

Cleaning features (88006, 16)
done.

    After (METRIC_NAME)
0     BufferCacheHitRatio
1     BufferCacheHitRatio
2     BufferCacheHitRatio
3     BufferCacheHitRatio
Name: METRIC_NAME, dtype: object

Done.
```

In [11]:
```
# Load and Cleanup Snapshot Data

def loadAndCleanSnapshot(csvLoc_in, verbose_in):

    if verbose_in:
        print("LoadAndCleanSnapshot")
        print("   " + str(csvLoc_in))

    mySnapDF = loadData(csvLoc_in,True)

    if verbose_in:
        print("   BEGIN mySnapDF columns:", mySnapDF.columns.tolist())

    # Don't need to use standard cleanupFeatures function
    #mySnapDF = cleanupFeatures(mySnapDF, [mySnapDF.columns], True)

    # Adding new date/time features for easier date display
    #snapDF['snap_dur_sec'] = (snapDF['end_interval_time'].astype('dat
    mySnapDF['beg_time']     = mySnapDF['BEGIN_INTERVAL_TIME']
    mySnapDF['end_time']     = mySnapDF['END_INTERVAL_TIME']
    mySnapDF['snap_id']      = mySnapDF['SNAP_ID']

    features = ['snap_id','beg_time','end_time']
    mySnapDF   = mySnapDF[features]

    if verbose_in:
        print()
        print("   AFTER mySnapDF columns:", mySnapDF.columns.tolist())
        print("done.")

    return(mySnapDF)


if testing:
    snapDF = loadAndCleanSnapshot("dmwprefenvsnpdet.csv",True)

print("\nDone.")
```

```
LoadAndCleanSnapshot
   dmwprefenvsnpdet.csv
Loading data
   dmwprefenvsnpdet.csv
   Loading data from local machine...done.
   Shape (1506, 11)
   BEGIN mySnapDF columns: ['SNAP_ID', 'DBID', 'INSTANCE_NUMBER', 'ST
ARTUP_TIME', 'BEGIN_INTERVAL_TIME', 'END_INTERVAL_TIME', 'FLUSH_ELAPS
ED', 'SNAP_LEVEL', 'ERROR_COUNT', 'SNAP_FLAG', 'SNAP_TIMEZONE']

   AFTER mySnapDF columns: ['snap_id', 'beg_time', 'end_time']
```

```
done.

Done.
```

In [12]:

```
# Denormalize. Focused on dba_hist_sysmetric_summary data

def denormalize(df_in,verbose_in):

    if verbose_in:
        print("Denormalizing")
        print("   BEFORE ", df_in.shape)

    df_inPiv = df_in.pivot_table(index='SNAP_ID', values='AVERAGE', co
    df_inPiv.reset_index(inplace=True)

    if verbose_in:
        print("   AFTER  ", df_inPiv.shape)
        print("done.")

    return(df_inPiv)


if testing:
    print("Testing Function: denormalize")
    features = ['SNAP_ID','METRIC_NAME','AVERAGE']
    testDF = testDF[features]
    testDFpiv = denormalize(testDF, True)
    print()
    print(testDFpiv[['SNAP_ID','AverageActiveSessions', 'CPUUsagePerSe

print("\nDone.")
```

```
Testing Function: denormalize
Denormalizing
   BEFORE  (88006, 3)
   AFTER   (280, 159)
done.

METRIC_NAME  SNAP_ID  AverageActiveSessions  CPUUsagePerSec  MemorySo
rtsRatio
0               4902               0.083737        6.787013         9
9.997501
1               4903               0.069534        6.485955        10
0.000000
2               4904               0.077124        6.815617        10
0.000000
3               4905               0.069264        6.431147        10
0.000000
4               4906               0.069193        6.469146        10
0.000000

Done.
```

In [13]:

```
# denormalizeWaitStats. Focused on dba_hist_WAITSTATS data

def denormalizeWaitStats(df_in,verbose_in):

    if verbose_in:
        print("Denormalizing")
        print("   BEFORE ", df_in.shape)

    df_inPiv = df_in.pivot_table(index='SNAP_ID', values='WAIT_COUNT',
    df_inPiv.reset_index(inplace=True)

    if verbose_in:
        print("   AFTER  ", df_inPiv.shape)
        print("done.")

    return(df_inPiv)


if testing:
    print("Testing Function: loadData")
    testDF1 = loadData("dmeperfenv_waitevent.csv",True)

    print("Testing Function: denormalize")
    features = ['SNAP_ID','CLASS','WAIT_COUNT']
    testDF1 = testDF1[features]
    testDFpiv1 = denormalizeWaitStats(testDF1, True)
    print()
    print(testDFpiv1[['SNAP_ID','save undo block', 'data block','unuse
    #print(testDFpiv.columns)

print("\nDone.")
```

```
Testing Function: loadData
Loading data
   dmeperfenv_waitevent.csv
   Loading data from local machine...done.
   Shape (25848, 6)
Testing Function: denormalize
Denormalizing
   BEFORE  (25848, 3)
   AFTER   (718, 19)
done.

CLASS  SNAP_ID  save undo block  data block  unused
0         4937              0.0     23487.5     0.0
1         4938              0.0     23618.5     0.0
2         4939              0.0     23804.0     0.0
3         4940              0.0     23950.5     0.0
4         4941              0.0     57745.0     0.0
```

```
Done.
```

In [14]:

```
# Function: Dimension Reduction Using Either:
#           PCA: Principle Component Analysis
#           ICA: Independent Component Analysis

def DimReduce(df_in, model_in, dimensions_in, verbose_in):

    if verbose_in:
        print("Reducing dimensionality")

    in_shape = df_in.shape

    if model_in == 'PCA':

        from sklearn.decomposition import PCA                 # load l
        pca       = PCA(n_components=dimensions_in)           # init m
        array_out = pca.fit_transform(df_in)                  # fit DF

    elif model_in == 'ICA':

        from sklearn.decomposition import FastICA
        ICA       = FastICA(n_components=dimensions_in, random_state=1
        array_out = ICA.fit_transform(df_in)

    else:
        print("   ERROR Function DimReduce. Invalid model provide.")

    df_out = pd.DataFrame(data = array_out) # create DF from array

    if verbose_in:
        print("   From/to", df_in.shape, df_out.shape)

    return(df_out, array_out)

if testing:
    print("Testing Function: DimRed2")
    print("     BEFORE", testDF.shape)
    qaDF, bogus = DimReduce(testDFpiv.drop(columns=['SNAP_ID']), 'PCA'
    print("     AFTER ", qaDF.shape)
    print()
    print(qaDF.head(4))

print("\nDone.")
```

```
Testing Function: DimRed2
     BEFORE (88006, 3)
Reducing dimensionality
```

```
   From/to (280, 158) (280, 3)
     AFTER  (280, 3)

              0             1            2
0 -4.510193e+09  1.657342e+09  2.553002e+07
1 -4.605250e+09  1.658850e+09  2.489804e+07
2 -4.601535e+09  1.654584e+09  3.982954e+07
3 -4.605258e+09  1.661445e+09  2.505973e+07

Done.
```

In [15]:

```
# Function: Standardize: Center (mean=0) and Scale (stdev=1):
#           1. fit Standardize model with the given dataframe
#           2. transforms the given dataframe
#           4. returns the transformed data as Dataframe

def CS_encode(df_in, verbose_in):

    if verbose_in:
        print("Standardizing", end=" ")

    from sklearn.preprocessing import StandardScaler   # load lib
    scaler = preprocessing.StandardScaler().fit(df_in) # init and fit
    ARcs = scaler.transform(df_in)                     # scale/transfc
    DFcs = pd.DataFrame(ARcs, columns=df_in.columns)   # convert resul

    if verbose_in:
        print("done.")

    return(DFcs)                                       # return the c&


if testing:
    print("Testing Function: CS_encode\n")
    print(testDFpiv[['AverageActiveSessions','CPUUsagePerSec']].descri
    print()

    qaDF = CS_encode(testDFpiv.drop(columns=['SNAP_ID']), True)

    print()
    print(qaDF[['AverageActiveSessions','CPUUsagePerSec']].describe())

print("\nDone.")
```

```
Testing Function: CS_encode

METRIC_NAME  AverageActiveSessions  CPUUsagePerSec
count                   280.000000      280.000000
mean                     12.306705      321.242847
```

```
std                        42.866774      661.166128
min                         0.002932        0.306437
25%                         0.110068        8.995016
50%                         1.078019       68.091407
75%                         4.535926      310.611356
max                       407.750755     4102.342859

Standardizing done.

METRIC_NAME  AverageActiveSessions  CPUUsagePerSec
count                 2.800000e+02    2.800000e+02
mean                 -2.061843e-17    2.307678e-16
std                   1.001791e+00    1.001791e+00
min                  -2.875374e-01   -4.862788e-01
25%                  -2.850337e-01   -4.731139e-01
50%                  -2.624128e-01   -3.835718e-01
75%                  -1.816020e-01   -1.610870e-02
max                   9.241472e+00    5.729075e+00

Done.
```

In [16]:

```
# Function: From a dataframe, create cluster,
# returning the model def, fitted model and fitted predict model

def create_cluster(df_in, cluster_type_in, cluster_no_in, verbose_in):

    if verbose_in:
        print("Creating cluster " + str(cluster_type_in) + " " + str(c

    if cluster_type_in == 'kmeans':

        from sklearn.cluster import KMeans
        mymodel          = KMeans(n_clusters=cluster_no_in, init='k-me
        mymodelfit       = mymodel.fit(df_in)
        mymodelfitlabels = mymodelfit.labels_
        mymodelfitpred   = mymodelfit.predict(df_in)

    elif cluster_type_in == 'MiniBatchKMeans':

        from sklearn.cluster import MiniBatchKMeans
        mymodel          = MiniBatchKMeans(n_clusters=cluster_no_in, b
        mymodelfit       = mymodel.fit(df_in)
        mymodelfitlabels = mymodelfit.labels_
        mymodelfitpred   = mymodelfit.predict(df_in)

    else:
        print("   ERROR in function, create_cluster")

    if verbose_in:
            print("done.")

        return(mymodel, mymodelfit, mymodelfitpred)


if testing:

    print("Testing Function: create_cluster")

    modelList = ['kmeans','MiniBatchKMeans']

    for myclustertype in modelList:

        print()
        print(myclustertype + str("...................................
        myclusterno   = 1
        qaDF = CS_encode(testDFpiv.drop(columns=['SNAP_ID']), True)
        myModel, myModelFit, myModelFitPred = create_cluster(qaDF, myc
        print("")
        print("   Cluster type        :", myclustertype)
        print("   Cluster numbers     :", myclusterno)
        print("   Cluster points      :", len(myModelFitPred))
        #print("   Counter Fit Labels  :", Counter(myModelFit.labels_)
        print("   Counter Fit U Labels:", unique(myModelFitPred))
        print("   Cluster model       :", myModel)

print("\nDone.")
```

```
Testing Function: create_cluster

kmeans.......................................
Standardizing done.
Creating cluster kmeans 1 done.

   Cluster type        : kmeans
   Cluster numbers     : 1
   Cluster points      : 280
   Counter Fit U Labels: [0]
   Cluster model       : KMeans(n_clusters=1, random_state=0)

MiniBatchKMeans.......................................
Standardizing done.
Creating cluster MiniBatchKMeans 1 done.

   Cluster type        : MiniBatchKMeans
   Cluster numbers     : 1
   Cluster points      : 280
   Counter Fit U Labels: [0]
   Cluster model       : MiniBatchKMeans(max_iter=300, n_clusters=1,
n_init=10, random_state=0)

Done.
```

In [17]:

```
# Function: Calculate distances between the a given cluster center and
#           every point in the given Dataframe.
#           AND, determine the anomaly threshold value

def get_point_to_centroid(cluster_type_in, cluster_init_in, cluster_fi

    if verbose_in:
        print("Get_point_to_centroid")

    from numpy import linalg as LA

    mypoints = points_DF_in.to_numpy()
    distances=[]
    i = 0
    for datapoint in mypoints:
        #print(datapoint)
        distances.append( LA.norm(datapoint-cluster_init_in.cluster_ce

        i = i +1

    points_DF_out              = points_DF_in
    points_DF_out['distance']  = distances

    threshold                  = np.quantile(distances, 0.98)  # super

    points_DF_out['threshold'] = threshold

    if verbose_in:
        # Calculate statistics, choose and set threshold value
        print("     mean=%0.2f median=%0.2f" % ( np.mean(distances),
        print("     95-pct=%0.2f 98-pct=%0.2f" % ( np.quantile(distar
        print("     min=%0.2f max=%0.2f" % ( np.min(distances), np.ma
        print("     points, threshold", len(distances), threshold)

    if verbose_in:
        print("done.")

    return(points_DF_out, threshold)


if testing:
    print("Testing Function: get_point_to_centroid")

    modelList = ['kmeans','MiniBatchKMeans']
    for myclustertype in modelList:

        print("\n" + myclustertype + str("...........................
        myclusters    = 1 # number of clusters created
        myclusterNo   = 0 # cluster number to get point centroid detai
        qaDF, bogus = DimReduce(testDFpiv.drop(columns=['SNAP_ID']), '
        qaDF = CS_encode(qaDF, True)
        myModel, myModelFit, myCluster = create_cluster(qaDF, mycluste
        print()
        print("      Cluster type   :", myclustertype)
        print("      Cluster numbers:", myclusters)
        print("      Cluster points :", len(myCluster))
        print("      Cluster model  :", myModel)
        print("details:",myModelFit)
        mypointsDF, threshold = get_point_to_centroid(myclustertype, m
        print("      points, threshold",len(mypointsDF), threshold)
        print(mypointsDF.head(4))

    print("\nDone Testing Function: get_point_to_centroid")

print("\nDone.")
```

```
Testing Function: get_point_to_centroid

kmeans.......................................

Reducing dimensionality
   From/to (280, 158) (280, 3)
Standardizing done.
Creating cluster kmeans 1 done.

      Cluster type   : kmeans
      Cluster numbers: 1
      Cluster points : 280
      Cluster model  : KMeans(n_clusters=1, random_state=0)
details: KMeans(n_clusters=1, random_state=0)
Get_point_to_centroid
      mean=1.43 median=1.16
      95-pct=3.37 98-pct=4.61
      min=0.07 max=7.30
      points, threshold 280 4.608793798454718
done.
      points, threshold 280 4.608793798454718
          0         1         2  distance  threshold
0 -0.472917  2.060596  0.093398  2.116230   4.608794
1 -0.482884  2.062471  0.091086  2.120203   4.608794
2 -0.482495  2.057167  0.145710  2.118010   4.608794
3 -0.482885  2.065697  0.091677  2.123367   4.608794

MiniBatchKMeans
```

```
MiniBatchKMeans.......................................


Reducing dimensionality
   From/to (280, 158) (280, 3)
Standardizing done.
Creating cluster MiniBatchKMeans 1 done.

      Cluster type   : MiniBatchKMeans
      Cluster numbers: 1
      Cluster points : 280
      Cluster model  : MiniBatchKMeans(max_iter=300, n_clusters=1, n_
init=10, random_state=0)
details: MiniBatchKMeans(max_iter=300, n_clusters=1, n_init=10, rando
m_state=0)
Get_point_to_centroid
      mean=1.44 median=1.17
      95-pct=3.37 98-pct=4.59
      min=0.08 max=7.28
      points, threshold 280 4.592654424989848
done.
      points, threshold 280 4.592654424989848
          0         1         2  distance  threshold
0 -0.472917  2.060596  0.093398  2.121590   4.592654
1 -0.482884  2.062471  0.091086  2.125665   4.592654
2 -0.482495  2.057167  0.145710  2.122889   4.592654
3 -0.482885  2.065697  0.091677  2.128817   4.592654

Done Testing Function: get_point_to_centroid

Done.
```

In [18]:

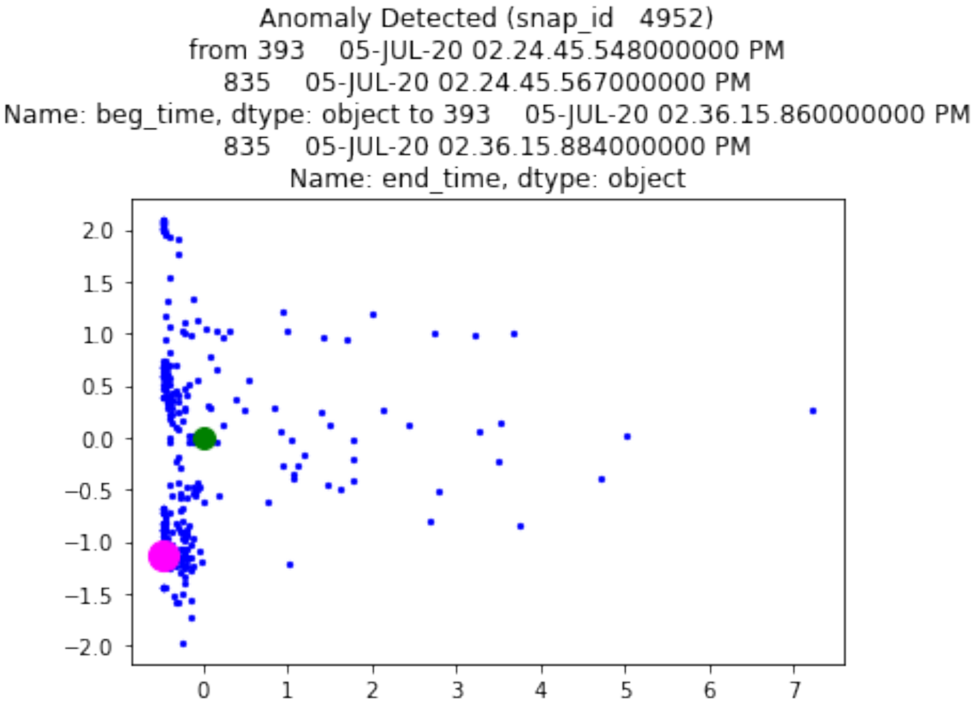
```
# Function: Return the begin and time times for a given snap_id

def get_b_e_times(snapDF_in, snap_id_in):

    b_time      = (snapDF_in['beg_time'].loc[snapDF_in['snap_id'] == s
    e_time      = (snapDF_in['end_time'].loc[snapDF_in['snap_id'] == s

    return(b_time, e_time)

if testing:
    print("Testing Function: get_b_e_times")
    print(snapDF.shape)
    print(snapDF.head(2))

    print()
    print("    First snap_id  ", snapDF['snap_id'][0])
    print("    First begin/end", get_b_e_times(snapDF, snapDF['snap_ic
    print("\nDone Testing function: get_b_e_times")

print("\nDone.")
```

```
Testing Function: get_b_e_times
(1506, 3)
   snap_id                         beg_time                         e
nd_time
0     4978  06-JUL-20 03.00.01.316000000 PM  06-JUL-20 04.00.04.65700
0000 PM
1     4984  06-JUL-20 09.00.13.132000000 PM  06-JUL-20 10.00.25.13700
0000 PM

    First snap_id   4978
    First begin/end (0       06-JUL-20 03.00.01.316000000 PM
1011    06-JUL-20 03.00.01.351000000 PM
Name: beg_time, dtype: object, 0       06-JUL-20 04.00.04.657000000 P
M
1011    06-JUL-20 04.00.04.694000000 PM
Name: end_time, dtype: object)

Done Testing function: get_b_e_times

Done.
```

In [19]:

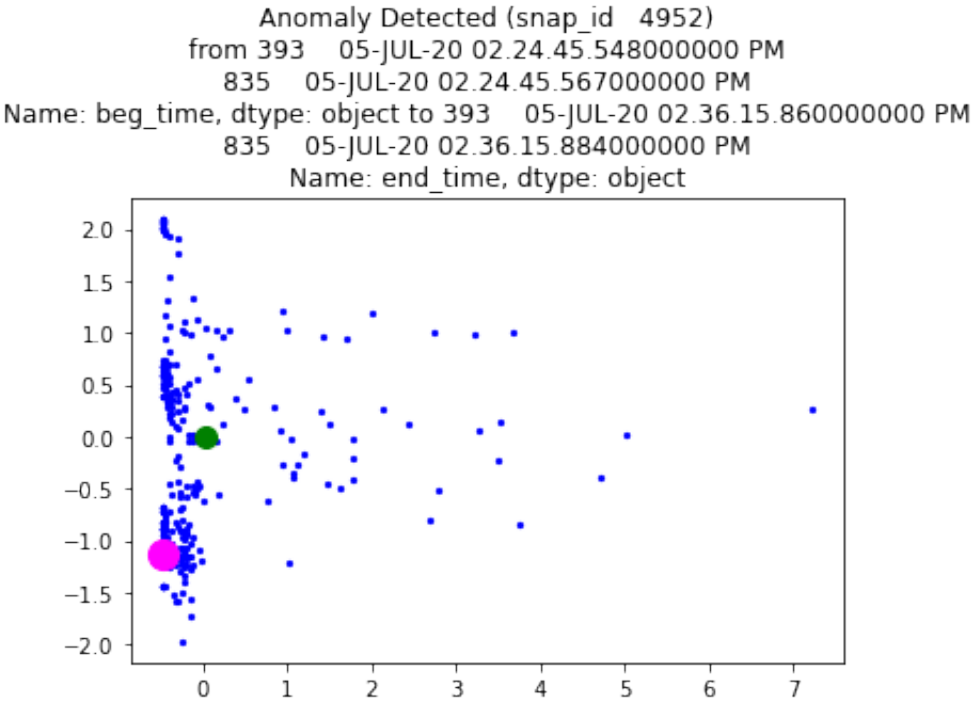
```
# Function: Print anomalous chart (screen and disk)
#           Assumption is an anomalous situation has been detected

def chart_anom2D(featuresDF_in, snapDF_in, snap_id_in, cluster_type_ir

    # To plot a 2D chart, must reduce dimensnions from N to 2.
    featuresDF, features = DimReduce(featuresDF_in.drop(columns=['SNAP

    if CS_in:
        featuresDF = CS_encode(featuresDF, True)
        features   = featuresDF.to_numpy()

    # Create a 1 cluster model
    myclustertype = cluster_type_in
    myclusters    = 1 # number of clusters created
    myclusterNo   = 0 # cluster number to get point centroid details,
    myModel, myModelFit, myCluster = create_cluster(featuresDF, myclus

    # Plot all points, including the current snap_id point
    plt.scatter(features[:,0], features[:,1], s=5, c='blue')      # us
    #plt.scatter(featuresDF.C1, featuresDF.C2, s=5, c='blue')   # usin

    # Plot the cluster center(s), only 1 for LVC
    plt.scatter(myModel.cluster_centers_[:, 0], myModel.cluster_center

    # Plot the most recent point/snap
    rowidx = len(featuresDF.index)-1
    #print("rowidx",rowidx)
    plt.scatter(features[rowidx,0], features[rowidx,1], s=200, c='mage

    # Set the title
    b_time, e_time = get_b_e_times(snapDF_in, snap_id_in)
    mytitle1 = "Anomaly Detected (snap_id {s:6d})\n".format(s=snap_id_
    mytitle2 = "from {beg} to {end}".format(beg=b_time, end=e_time)
    plt.title(mytitle1+mytitle2)

    # Save chart to disk in the existing sub directories
    filename2 = 'pypics/' + str(cluster_type_in) + '/' + str(snap_id_
    plt.savefig(filename2)

    # Display the chart
    plt.show()

if testing:

    print("Testing Function: chart_anom")
    testrowidx = 50
    the_snap_id = testDFpiv['SNAP_ID'][testrowidx]

    for testcluster_type in ['kmeans','MiniBatchKMeans']:
        print("---------- Cluster type ", testcluster_type)
        chart_anom2D(testDFpiv, snapDF, the_snap_id, testcluster_type,

    print("Done Testing Function: chart_anom")

print("\nDone.")
```

```
Testing Function: chart_anom
---------- Cluster type  kmeans
Standardizing done.
Creating cluster kmeans 1 done.
```



Anomaly Detected (snap_id 4952)
from 393 05-JUL-20 02.24.45.548000000 PM
835 05-JUL-20 02.24.45.567000000 PM
Name: beg_time, dtype: object to 393 05-JUL-20 02.36.15.860000000 PM
835 05-JUL-20 02.36.15.884000000 PM
Name: end_time, dtype: object

```
---------- Cluster type  MiniBatchKMeans
Standardizing done.
Creating cluster MiniBatchKMeans 1 done.
```



Anomaly Detected (snap_id 4952)
from 393 05-JUL-20 02.24.45.548000000 PM
835 05-JUL-20 02.24.45.567000000 PM
Name: beg_time, dtype: object to 393 05-JUL-20 02.36.15.860000000 PM
835 05-JUL-20 02.36.15.884000000 PM
Name: end_time, dtype: object

```
Done Testing Function: chart_anom

Done.
```

Create Model, Detect Anomaly And Alert If Necessary

A single cluster unsupervised K-means algo model is created and anomolous snaps are identified. If the most recent snap is identified as anomalous, an alert is triggered (different color and size point, plot title is different).

A similar process can be implemented in a production environment, to constanly look for anomalous performance activity, based on the most recent snap activity.

In [15]:

```
print("Checking for anomalous activity now...\n")

cluster_type = 'kmeans'  # kmeans, MiniBatchKMeans
doCS         = True      # True: Do Standarize, False: Do NOT Standard

sysmetricDF    = loadData("dmwperfenv.csv",True)
sysmetricDF    = cleanupFeatures(sysmetricDF, ['METRIC_NAME'], True)
sysmetricDFpiv = denormalize(sysmetricDF, True)

print()

snapDF = loadAndCleanSnapshot("dmwprefenvsnpdet.csv",True)

print("\nDone.")
```

```
Checking for anomalous activity now...

Loading data
   dmwperfenv.csv
   Loading data from local machine...done.
   Shape (88006, 16)
Cleaning features (88006, 16)
done.
Denormalizing
   BEFORE  (88006, 16)
   AFTER   (280, 159)
done.

LoadAndCleanSnapshot
   dmwprefenvsnpdet.csv
Loading data
   dmwprefenvsnpdet.csv
   Loading data from local machine...done.
   Shape (1506, 11)
   BEGIN mySnapDF columns: ['SNAP_ID', 'DBID', 'INSTANCE_NUMBER', 'ST
ARTUP_TIME', 'BEGIN_INTERVAL_TIME', 'END_INTERVAL_TIME', 'FLUSH_ELAPS
ED', 'SNAP_LEVEL', 'ERROR_COUNT', 'SNAP_FLAG', 'SNAP_TIMEZONE']

   AFTER mySnapDF columns: ['snap_id', 'beg_time', 'end_time']
done.

Done.
```

In [21]:

```
print("Checking for anomalous activity now...\n")

cluster_type = 'kmeans'  # kmeans, MiniBatchKMeans
doCS         = False      # True: Do Standarize, False: Do NOT Standar

sysmetricDF    = loadData("dmeperfenv_waitevent.csv",True)
sysmetricDF    = cleanupFeatures(sysmetricDF, ['CLASS'], True)
sysmetricDFpiv = denormalizeWaitStats(sysmetricDF, True)

print()

snapDF = loadAndCleanSnapshot("dmwprefenvsnpdet.csv",True)

print("\nDone.")
```

```
Checking for anomalous activity now...

Loading data
   dmeperfenv_waitevent.csv
   Loading data from local machine...done.
   Shape (25848, 6)
Cleaning features (25848, 6)
done.
Denormalizing
   BEFORE  (25848, 6)
   AFTER   (718, 19)
done.

LoadAndCleanSnapshot
   dmwprefenvsnpdet.csv
Loading data
   dmwprefenvsnpdet.csv
   Loading data from local machine...done.
   Shape (1506, 11)
   BEGIN mySnapDF columns: ['SNAP_ID', 'DBID', 'INSTANCE_NUMBER', 'ST
ARTUP_TIME', 'BEGIN_INTERVAL_TIME', 'END_INTERVAL_TIME', 'FLUSH_ELAPS
ED', 'SNAP_LEVEL', 'ERROR_COUNT', 'SNAP_FLAG', 'SNAP_TIMEZONE']

   AFTER mySnapDF columns: ['snap_id', 'beg_time', 'end_time']
done.

Done.
```

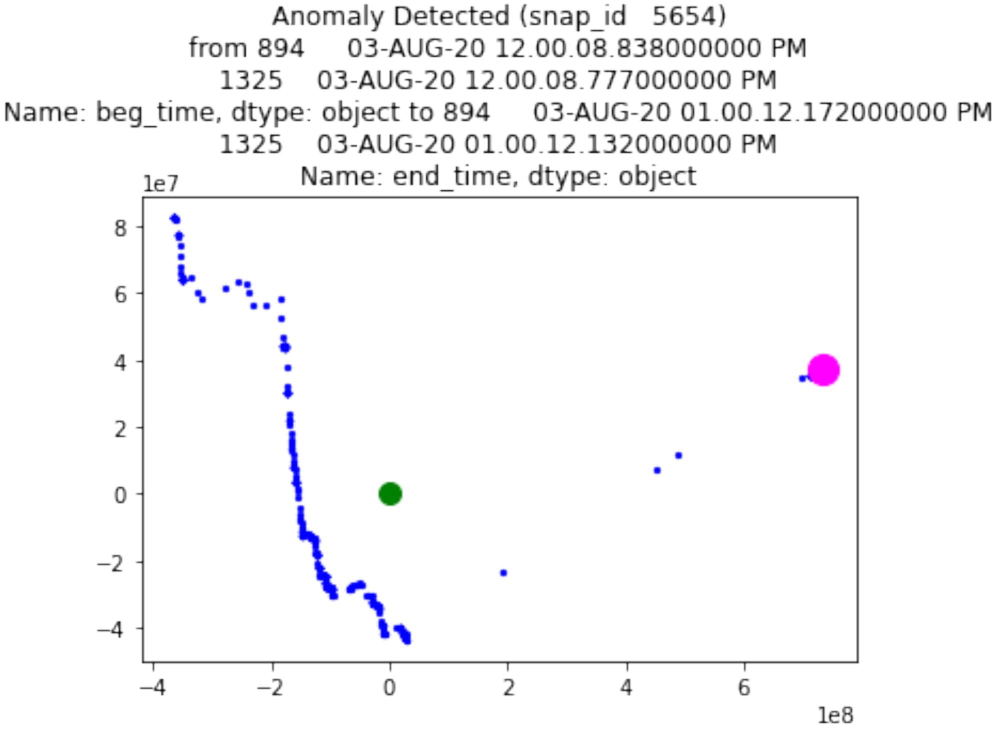
In [22]:

```
# FOR DEPLOYMENT

# For deployment, build the cluster with ALL available data and
# check if the most recent snap is anomalous.

endidx   = len(sysmetricDFpiv.index)+1
```

```
#
# 1. Build The Cluster - from row 0 to row endidx
#

workDF = sysmetricDFpiv[0:endidx]

beginSnapID = int(workDF.head(1)['SNAP_ID'].values)
endSnapID   = int(workDF.tail(1)['SNAP_ID'].values)

print("Building cluster from/to snap_id {sb:4d}/{se:4d}, {l:4d} snap_i

if doCS:
    workDF = CS_encode(workDF, False)

model_init, model_fit, model_fitpred = create_cluster(workDF.drop(colu

#
# 2. Determine Anomaly Distance Threshold
#

distancesDF, threshold = get_point_to_centroid(cluster_type, model_ini

currentDistance  = float(distancesDF.tail(1)['distance'].values)
currentThreshold = float(distancesDF.tail(1)['threshold'].values)
#currentSnapID   = endSnapID

#
# 3. Check If Most Recent Snap Is An Anomaly. If so, alert...
#

if currentDistance > currentThreshold:

    #
    # 4. Anomaly detected... Alert!
    #

    b_time, e_time = get_b_e_times(snapDF, endSnapID)

    print("* Anomoly detected for snap_id {s:6d}".format(s=endSnapID))
    print("     from {beg} to {end}".format(beg=b_time, end=e_time))
    print("     dist={d:8.3f} > thresh={t:8.3f} loop={loop:8d}".format
    chart_anom2D(workDF, snapDF, endSnapID, cluster_type, CS_in=doCS)

    # Quick! Alert the DBAs!!

else:
    print("     Anomoly NOT detected for snap_id {s:6d}  (distance={d:
```

```
print("\nDone checking.\n")
```

```
Building cluster from/to snap_id 4937/5654,  719 snap_ids. Checkin
g...
* Anomoly detected for snap_id   5654
     from 894     03-AUG-20 12.00.08.838000000 PM
1325    03-AUG-20 12.00.08.777000000 PM
Name: beg_time, dtype: object to 894     03-AUG-20 01.00.12.172000000
PM
1325    03-AUG-20 01.00.12.132000000 PM
Name: end_time, dtype: object
     dist=736729242.718 > thresh=723511920.347 loop=     719
Creating cluster kmeans 1 done.
```



```
Done checking.
```

In [18]:

```
#look in past data
# Simply overwriting above startidx and endidx variables)
startidx = 450    # S1:  5   S2: 500
endidx   = 500   # S1: 50  S2: 800

for rowidx in range(startidx, endidx):

    #
    # 1. Build The Cluster - from row 0 to row rowidx. First rowidx wi
    #
    workDF = sysmetricDFpiv[0:rowidx]

    beginSnapID = int(workDF.head(1)['SNAP_ID'].values)
    endSnapID   = int(workDF.tail(1)['SNAP_ID'].values)

    print("Building cluster from/to snap_id {sb:4d}/{se:4d} loop {l:4d

    if doCS:
        workDF = CS_encode(workDF, False)

    model_init, model_fit, model_fitpred = create_cluster(workDF.drop(

    #
    # 2. Determine Anomaly Distance Threshold
    #

    distancesDF, threshold = get_point_to_centroid(cluster_type, model

    currentDistance  = float(distancesDF.tail(1)['distance'].values)
    currentThreshold = float(distancesDF.tail(1)['threshold'].values)
    #currentSnapID   = endSnapID

    #
    # 3. Check If Most Recent Snap Is An Anomaly. If so, alert...
    #

    if currentDistance > currentThreshold:

        #
        # 4. Anomaly detected... Alert!
        #

        b_time, e_time = get_b_e_times(snapDF, endSnapID)

        print("* Anomoly detected for snap_id {s:6d}".format(s=endSnap
        print("     from {beg} to {end}".format(beg=b_time, end=e_time
        print("     dist={d:8.3f} > thresh={t:8.3f} loop={loop:8d}".fo
        chart_anom2D(workDF, snapDF, endSnapID, cluster_type, CS_in=do

        # Quick! Alert the DBAs!!

    else:
        print("     Anomoly NOT detected for snap_id {s:6d}  (distance


print("\nDone checking.\n")
```

```
Building cluster from/to snap_id 4937/5386 loop  450 Checking...
     Anomoly NOT detected for snap_id   5386  (distance=162082075.002
<= threshold=212667591.473)
Building cluster from/to snap_id 4937/5387 loop  451 Checking...
     Anomoly NOT detected for snap_id   5387  (distance=163205814.645
<= threshold=213028641.882)
Building cluster from/to snap_id 4937/5388 loop  452 Checking...
     Anomoly NOT detected for snap_id   5388  (distance=163561117.688
<= threshold=213389708.461)
Building cluster from/to snap_id 4937/5389 loop  453 Checking...
     Anomoly NOT detected for snap_id   5389  (distance=163362337.664
<= threshold=213749543.128)
Building cluster from/to snap_id 4937/5390 loop  454 Checking...
     Anomoly NOT detected for snap_id   5390  (distance=163133863.215
<= threshold=214108088.135)
Building cluster from/to snap_id 4937/5391 loop  455 Checking...
     Anomoly NOT detected for snap_id   5391  (distance=163824113.223
<= threshold=214467311.916)
Building cluster from/to snap_id 4937/5392 loop  456 Checking...
     Anomoly NOT detected for snap_id   5392  (distance=163509991.331
<= threshold=214825061.055)
Building cluster from/to snap_id 4937/5393 loop  457 Checking...
     Anomoly NOT detected for snap_id   5393  (distance=182681410.052
<= threshold=215222529.031)
Building cluster from/to snap_id 4937/5394 loop  458 Checking...
     Anomoly NOT detected for snap_id   5394  (distance=188316161.644
<= threshold=215630944.618)
Building cluster from/to snap_id 4937/5395 loop  459 Checking...
     Anomoly NOT detected for snap_id   5395  (distance=187965566.777
<= threshold=216037724.467)
Building cluster from/to snap_id 4937/5396 loop  460 Checking...
     Anomoly NOT detected for snap_id   5396  (distance=187622143.301
<= threshold=216442888.461)
Building cluster from/to snap_id 4937/5397 loop  461 Checking...
     Anomoly NOT detected for snap_id   5397  (distance=187281098.012
<= threshold=216846448.710)
Building cluster from/to snap_id 4937/5398 loop  462 Checking...
     Anomoly NOT detected for snap_id   5398  (distance=186943563.081
<= threshold=217248419.393)
Building cluster from/to snap_id 4937/5399 loop  463 Checking...
     Anomoly NOT detected for snap_id   5399  (distance=186711687.610
<= threshold=217649023.320)
Building cluster from/to snap_id 4937/5400 loop  464 Checking...
     Anomoly NOT detected for snap_id   5400  (distance=186435543.219
<= threshold=218048175.442)
Building cluster from/to snap_id 4937/5401 loop  465 Checking...
     Anomoly NOT detected for snap_id   5401  (distance=186047746.254
<= threshold=218445649.275)
Building cluster from/to snap_id 4937/5402 loop  466 Checking...
     Anomoly NOT detected for snap_id   5402  (distance=186253378.985
<= threshold=218842766.751)
Building cluster from/to snap_id 4937/5403 loop  467 Checking...
```

```
     Anomoly NOT detected for snap_id   5403  (distance=186195920.065
<= threshold=219238977.811)
Building cluster from/to snap_id 4937/5404 loop  468 Checking...
     Anomoly NOT detected for snap_id   5404  (distance=186061937.000
<= threshold=219634108.316)
Building cluster from/to snap_id 4937/5405 loop  469 Checking...
     Anomoly NOT detected for snap_id   5405  (distance=187162087.416
<= threshold=220030658.155)
Building cluster from/to snap_id 4937/5406 loop  470 Checking...
     Anomoly NOT detected for snap_id   5406  (distance=188061103.945
<= threshold=220428189.390)
Building cluster from/to snap_id 4937/5407 loop  471 Checking...
     Anomoly NOT detected for snap_id   5407  (distance=187906930.532
<= threshold=220824545.252)
Building cluster from/to snap_id 4937/5408 loop  472 Checking...
     Anomoly NOT detected for snap_id   5408  (distance=187813454.637
<= threshold=221219925.354)
Building cluster from/to snap_id 4937/5409 loop  473 Checking...
     Anomoly NOT detected for snap_id   5409  (distance=187525644.839
<= threshold=221613871.264)
Building cluster from/to snap_id 4937/5410 loop  474 Checking...
     Anomoly NOT detected for snap_id   5410  (distance=187216131.110
<= threshold=222006340.734)
Building cluster from/to snap_id 4937/5411 loop  475 Checking...
     Anomoly NOT detected for snap_id   5411  (distance=186840924.585
<= threshold=222397206.524)
Building cluster from/to snap_id 4937/5412 loop  476 Checking...
     Anomoly NOT detected for snap_id   5412  (distance=186448542.924
<= threshold=222786440.662)
Building cluster from/to snap_id 4937/5413 loop  477 Checking...
     Anomoly NOT detected for snap_id   5413  (distance=186057825.594
<= threshold=223174053.322)
Building cluster from/to snap_id 4937/5414 loop  478 Checking...
     Anomoly NOT detected for snap_id   5414  (distance=185668754.723
<= threshold=223560054.579)
Building cluster from/to snap_id 4937/5415 loop  479 Checking...
     Anomoly NOT detected for snap_id   5415  (distance=185281376.841
<= threshold=223944454.540)
Building cluster from/to snap_id 4937/5416 loop  480 Checking...
     Anomoly NOT detected for snap_id   5416  (distance=184895503.207
<= threshold=224327262.873)
Building cluster from/to snap_id 4937/5417 loop  481 Checking...
     Anomoly NOT detected for snap_id   5417  (distance=184511265.900
<= threshold=224708489.448)
Building cluster from/to snap_id 4937/5418 loop  482 Checking...
     Anomoly NOT detected for snap_id   5418  (distance=184128618.775
<= threshold=225088143.992)
Building cluster from/to snap_id 4937/5419 loop  483 Checking...
     Anomoly NOT detected for snap_id   5419  (distance=183747545.186
<= threshold=225466236.126)
```

```
Building cluster from/to snap_id 4937/5420 loop  484 Checking...
     Anomoly NOT detected for snap_id   5420  (distance=183368105.255
<= threshold=225842775.539)
Building cluster from/to snap_id 4937/5421 loop  485 Checking...
     Anomoly NOT detected for snap_id   5421  (distance=182990235.931
<= threshold=226217771.736)
Building cluster from/to snap_id 4937/5422 loop  486 Checking...
     Anomoly NOT detected for snap_id   5422  (distance=182613877.811
<= threshold=226591234.047)
Building cluster from/to snap_id 4937/5423 loop  487 Checking...
     Anomoly NOT detected for snap_id   5423  (distance=182239049.745
<= threshold=226963171.784)
Building cluster from/to snap_id 4937/5424 loop  488 Checking...
     Anomoly NOT detected for snap_id   5424  (distance=181865771.751
<= threshold=227333594.240)
Building cluster from/to snap_id 4937/5425 loop  489 Checking...
     Anomoly NOT detected for snap_id   5425  (distance=181494009.495
<= threshold=227702510.588)
Building cluster from/to snap_id 4937/5426 loop  490 Checking...
     Anomoly NOT detected for snap_id   5426  (distance=181123765.138
<= threshold=228069929.946)
Building cluster from/to snap_id 4937/5427 loop  491 Checking...
     Anomoly NOT detected for snap_id   5427  (distance=180755047.002
<= threshold=228435861.398)
Building cluster from/to snap_id 4937/5428 loop  492 Checking...
     Anomoly NOT detected for snap_id   5428  (distance=180387889.631
<= threshold=228800314.037)
Building cluster from/to snap_id 4937/5429 loop  493 Checking...
     Anomoly NOT detected for snap_id   5429  (distance=180022137.321
<= threshold=229163296.602)
Building cluster from/to snap_id 4937/5430 loop  494 Checking...
     Anomoly NOT detected for snap_id   5430  (distance=179657851.514
<= threshold=229524817.897)
Building cluster from/to snap_id 4937/5431 loop  495 Checking...
     Anomoly NOT detected for snap_id   5431  (distance=179295082.514
<= threshold=229884886.770)
Building cluster from/to snap_id 4937/5432 loop  496 Checking...
     Anomoly NOT detected for snap_id   5432  (distance=178933759.034
<= threshold=230243511.882)
Building cluster from/to snap_id 4937/5433 loop  497 Checking...
     Anomoly NOT detected for snap_id   5433  (distance=178573913.439
<= threshold=230600701.902)
Building cluster from/to snap_id 4937/5434 loop  498 Checking...
     Anomoly NOT detected for snap_id   5434  (distance=178215469.612
<= threshold=230956465.303)
Building cluster from/to snap_id 4937/5435 loop  499 Checking...
     Anomoly NOT detected for snap_id   5435  (distance=177858481.405
<= threshold=231310810.611)

Done checking.
```

In [18]:

```
now = datetime.now()
dt_string = now.strftime("%d-%b-%Y %H:%M:%S")
print("Done with entire notebook at", dt_string)
```

```
Done with entire notebook at 30-Nov-2020 15:09:28
```

In [ ]: