Arno Dunstatter, 1926976

Task 4

Subtask A

```
def PUR(predicted, actual, outliers=False): #here a = predicted, b = actual
  if len(predicted) != len(actual): # verify we have the same number of elements in each
    print("Error: Unequal number of elements in predicted and actual passed to PUR()")
    return

  def clustersClasses(predicted, actual):
    dictToReturn = dict.fromkeys(set(predicted).union(set(actual)))
    for i in range(len(predicted)):
      if dictToReturn[predicted[i]] == None:
        dictToReturn[predicted[i]] = [actual[i]]
      else:
        dictToReturn[predicted[i]].append(actual[i])
    return dictToReturn

  def sumCountsOfMajorityClassInEachCluster(clustersActuals):
    sum = 0
    for actuals in clustersActuals.values():
      maxCount = 0
      if actuals != None:
        for value in set(actuals):
          if actuals.count(value) > maxCount:
            maxCount = actuals.count(value)
      sum += maxCount
    return sum

  N = len(predicted)
  clustersActuals = clustersClasses(predicted, actual)

  if outliers:
    # do not consider outliers in purity calculation, return a vector: (purity,percentage_of_outliers)
    numOutliers = list(predicted).count(-1)#len([x for x in predicted if x == -1]) #-1 as a predicted class indicates an outlier in the return of a DBSCAN clustering
    N -= numOutliers
    if clustersActuals.get(-1) != None: # there are outliers, remove them with dict.pop()
      clustersActuals.pop(-1)
    sum = sumCountsOfMajorityClassInEachCluster(clustersActuals)
    return (float(sum/N), float(numOutliers/N))

  else: # no outliers to consider,  just returns a floting point number of the observed purity
    sum = sumCountsOfMajorityClassInEachCluster(clustersActuals)
    return float(sum/N)
```

Arno Dunstatter, 1926976

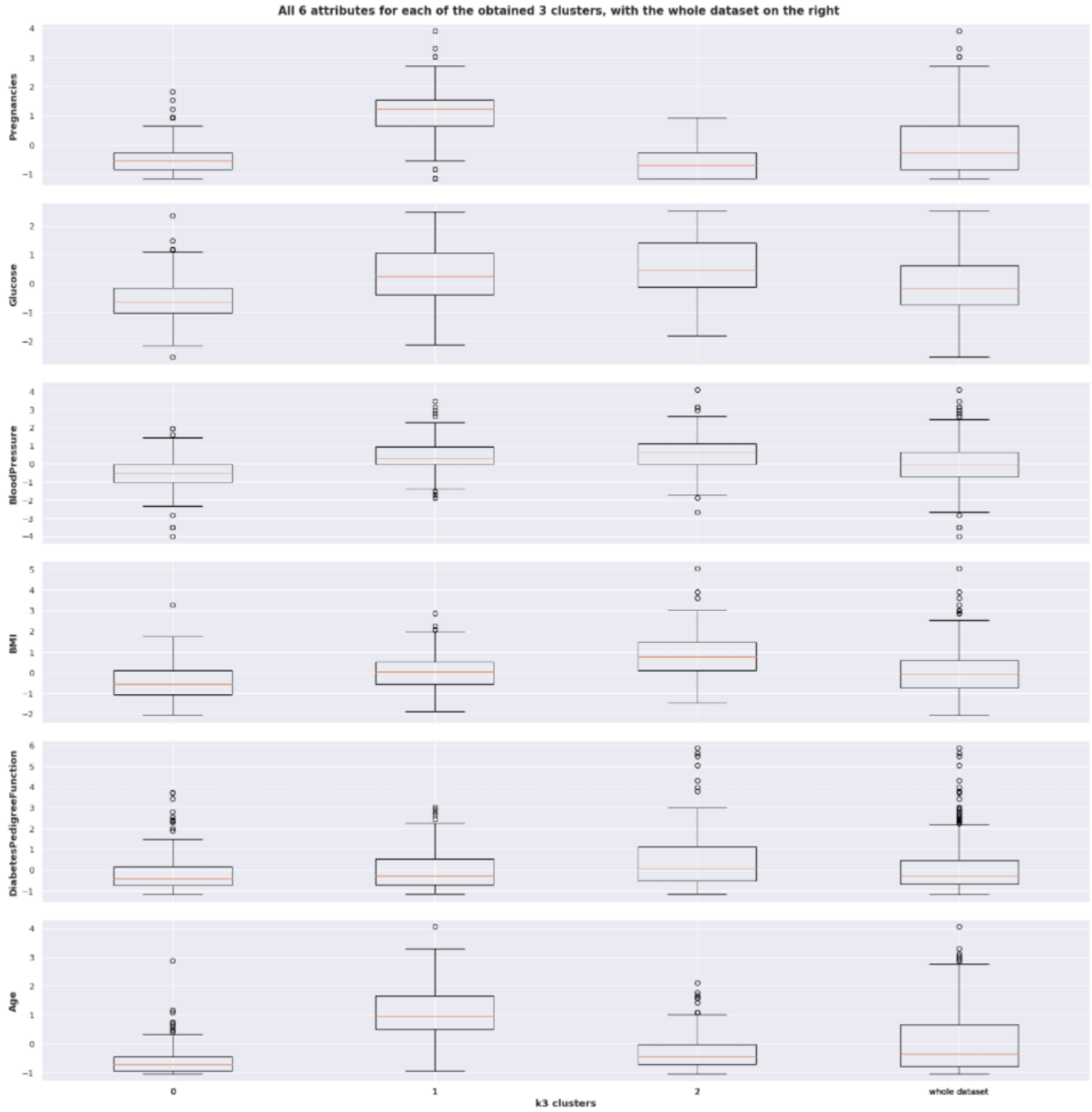
Subtask B

K-means was unable to “rediscover” the natural clusters of the Complex9 dataset. it's important to note that the KMeans algorithm won't actually discern the number of natural classes, it will simply place all datapoints, as best it can, into 1 of the k number of clusters the user tells it to. Even when we told it the correct number of clusters to place the items into, it still only achieved a ~72.4% purity in those clusters, indicating a substantial error between the classes it predicted for each datapoint and the actual class of that datapoint.

Arno Dunstatter, 1926976

Subtask C

The purity achieved by K-Means clustering of the Z-score normalized PID dataset, with k=3, was 67.71%.



Centroids

Pregnancies	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age	k3
-0.467313	-0.542295	-0.577479	-0.467030	-0.202561	-0.616597	0
1.132353	0.364154	0.396111	0.025493	-0.031080	1.136188	1
-0.551146	0.568430	0.594715	0.855952	0.424832	-0.272060	2

Based primarily on the boxplots comparing Glucose (but also on the comparisons of BMI, and Age) it appears that k=0 represents Outcome=0 (non-diabetic), while k=1 and k=2 represent Outcome=1 (diabetic).

Arno Dunstatter, 1926976

Subtask D

Performing DBSCAN on the ZPID dataset with epsilon=2.0 and min_samples=3 resulted in only 1.32% outliers and a purity of 65.30% was achieved. This fit resulted in two groups, one of which was an outlier group.

Arno Dunstatter, 1926976

Subtask E

The search procedure, bestDBSCAN():

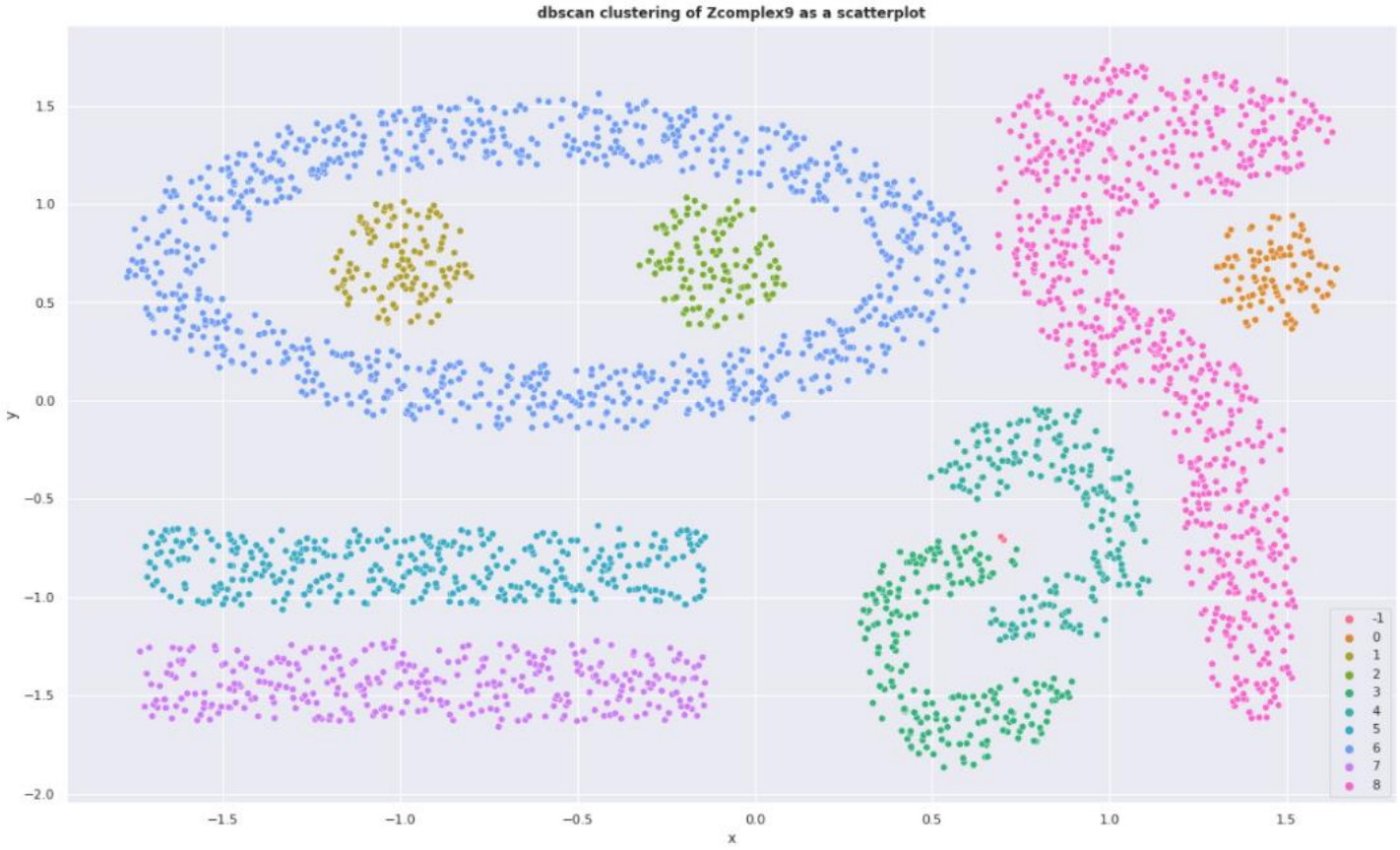
```
def bestDBSCAN(data, actual):
  goodParamsAndTheirResults = []
  count = 0
  eMin = 0.0001
  eStep = 0.05
  eCutOff = eMin+55*eStep
  mMin = 1
  mStep = 1
  mCutOff = mMin + 54*mStep
  for e in np.arange(eMin, eCutOff, eStep): # 55 iterations
    for m in range(mMin, mCutOff, mStep):  # 54 iterations, 55*54 = 2970 iterations
      fitObj = DBSCAN(eps=e,min_samples=m).fit(data)
      if len(set(fitObj.labels_))>=2 and len(set(fitObj.labels_))<=15: # 1. There should be between 2 and 15 clusters
        pur = PUR(fitObj.labels_, actual, outliers=True)
        if pur[1] <= 0.1: # 2. The number of outlirs should be 10% or less
          goodParamsAndTheirResults.append([fitObj,pur,e,m])
      count+=1
  print(f"{count} pairs of epsilon and minPoints were tested.")
  goodParamsAndTheirResults.sort(key=lambda x : (x[1][0],-x[1][1]), reverse=True) # after sorting the best will be at index=0
  #      bestDBSCANclusteringFound,              its purity,               all the others that had b/w 2 and 15 clusters and <= 10% outliers
  return goodParamsAndTheirResults[0][0], goodParamsAndTheirResults[0][1], goodParamsAndTheirResults
```

The bestDBSCAN(data,actual) function works by assuming that the data passed in has been Z-scored, that way the range for epsilon can begin from the same start point (if the data wasn't normalized then the scales of distances between points would be different depending on the dataset's attributes' scales). The procedure creates an empty list, goodParamsAndTheirResults, into which the procedure will place parameters, their resulting clustering, and the purity of that clustering, provided the stipulated specifications are met (between 2 and 15 clusters, and no more than 10% outliers). The procedure then uses two forloops, the outer-forloop which iterates over proposed values of epsilon (e), and the inner-forloop which iterates over proposed values of minSamples (m). Because of how these forloops' ranges were set up, they will iterate 2970 times. Inside the inner-most forloop a DBSCAN object is created and fit to the passed in data. If that clustering has between 2 and 15 clusters then its predicted labels and the actual, known labels are passed to the PUR() function for evaluation. If the % outliers is less than 10% then that cluster has met all specifications and it, its PUR results, and the parameters which produced it are added to the goodParamsAndTheirResults list. After completing the forloops the procedure then sorts the goodParamsAndTheirResults list such that highest purity clusters come first, prioritizing lower percentages of outliers whenever there are ties between purities. Finally the procedure returns 3 things:

1. The best clustering found
2. The best clustering's purity
3. The sorted list of all qualifiying clusterings, their purities, and their parameters

I Z-scored the Complex9 dataset before using the procedure, which iterated through 2970 pairs of between epsilon and minPoints. For epsilon=0.1001 and minPoints=8 only 0.07% were outliers and a purity of 100.00% was achieved. This fit resulted in 9 natural clusters and 1 outlier cluster containing only 2 data points. I am quite happy with the achieved maximum purity and its associated marginal outlier percentage.

Graphing this clustering as a scatterplot yields the following:



Extra Credit:

When used on the ZPID dataset, the best clustering utilized epsilon=1.4 and minPoints=2 and resulted in 8.94% outliers and a purity of 68.09%. That said the purity may not actually be a great representation of performance here, as this clustering resulted in 7 natural clusters and 1 outlier cluster. In the original PID dataset 65% of cases had outcomes of 0 (not diabetic) with the remaining 35% being 1 (diabetic). In the best clustering generated by the procedure 90% of cases were predicted to be in cluster 0, with the remaining 10% being predicted to be in clusters -1, 1, 2, 3, 4, 5 or 6.