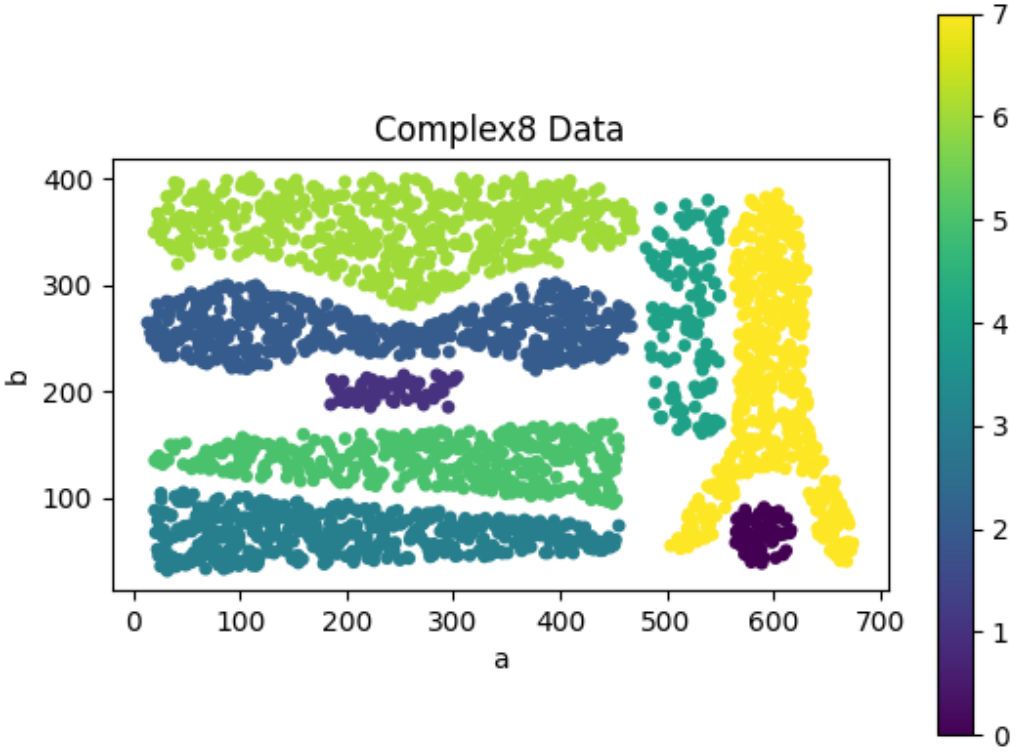
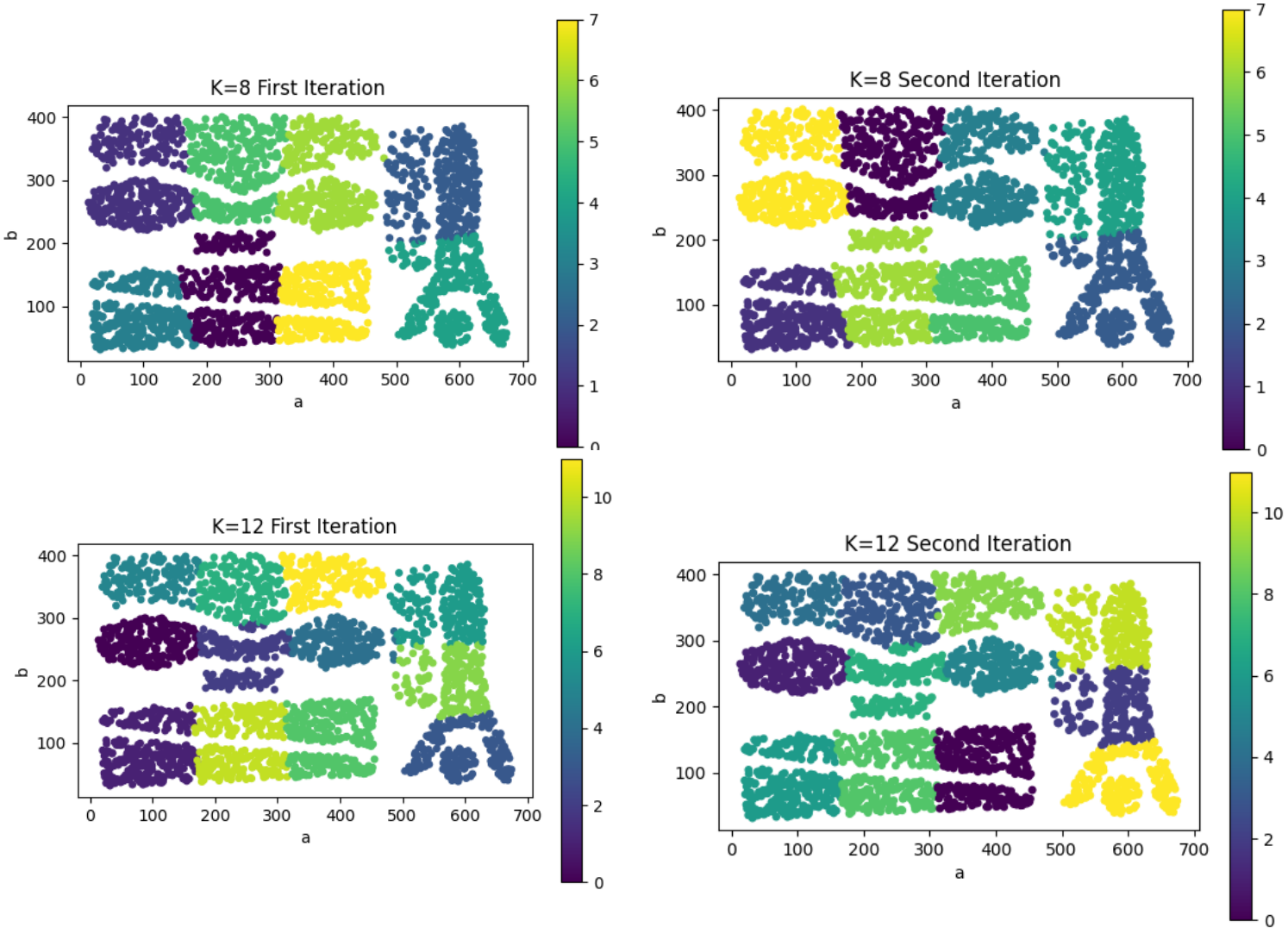
Roberto Jackson Baeza, Report

Subtask A



Given Purity with Outliers: (1.0, 0.02)
Given Purity without Outliers: 1.0

Subtask B

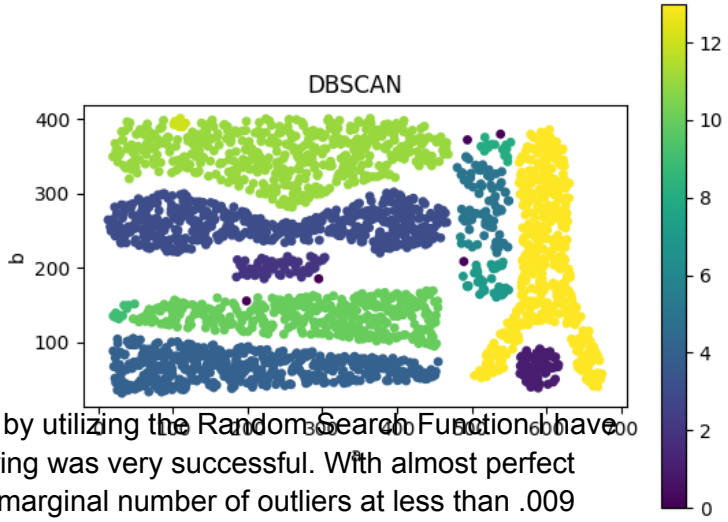


I called the K= function two times with different random state parameters. The first round has random state = 42 and the second round has random state = 99. Largely this only very slightly changes the data. For instance, the purity and percent outliers are only marginally different for k=8 and k=12. Overall the k=12 outperformed the k=8 k means classification. All of the classifications generally produced the same number of outliers around 0.10.

K	First Round Purity	Second Round Purity
8	Outliers: (0.64, 0.13). No Outliers: 0.78	Outliers: (0.64, 0.12). No Outliers: 0.64
12	Outliers: (0.78, 0.09). No Outliers: 0.78	Outliers: (0.78, 0.11). No Outliers: 0.78

Subtask C

MinPoints Range : [2, 4]
Epsilon Range: [13.6, 15.0]
Best MinPts: 2
Best Epsilon: 14.97117203926934
Number of Classes: 10
DBSCAN Purity with Outliers: (1.0, 0.0)



To the side is the Clustering result I got by utilizing the Random Search Function I have described below. Overall I think that the clustering was very successful. With almost perfect accuracy of over .99 Purity fraction and only a marginal number of outliers at less than .009 outliers fraction. The DBSCAN had Epsilon Distance of 14.97 and a Min Points Amount of 2. Every cluster but 2 is completely whole meanwhile two clusters are amalgamations.

Random Search Explanation

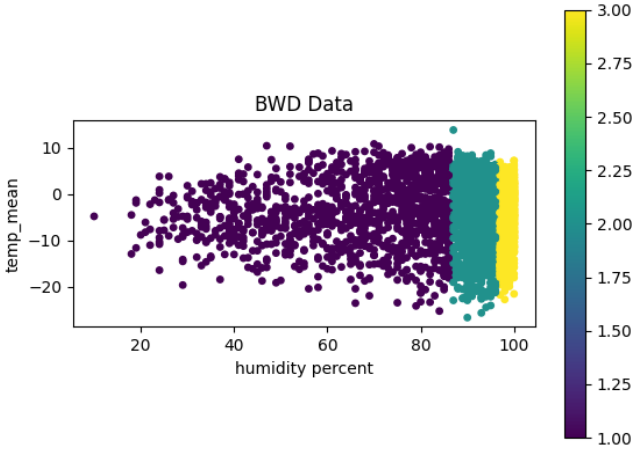
RandomSearch(data, groundTruth, minPointsRange, epsilonRange, maxIterations, file)

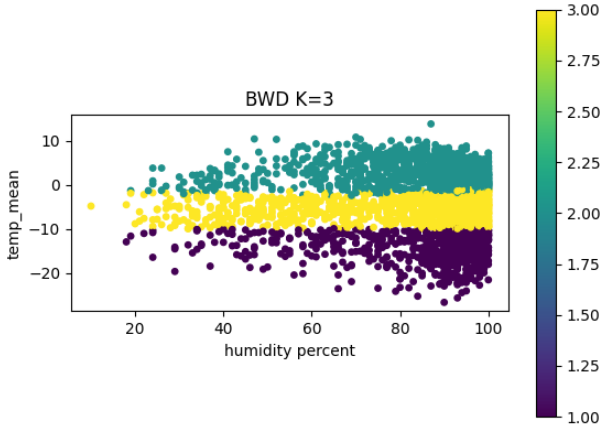
The way that my search algorithm works is by utilizing randomization, and optimization, through many iterations to generate a good enough method.

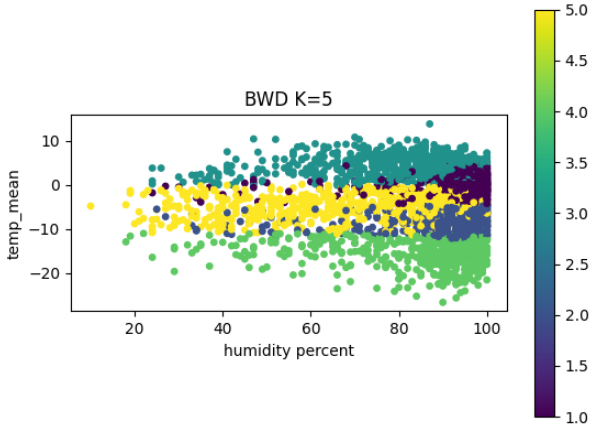
1. Initializes MaxPurity to 0, Min and Max clusters to their values, bestDBSCAN, best Esilon and best min points to 0
2. Creates a for loop that runs for max iterations
3. Randomly generate minpoints and epsilon from their respective ranges
4. Create a Dbscan with the minpoints and epsilon.
5. calculate the corresponding number of clusters, purity, and outliers
6. Check if purity is greater than maxPurity, and if clusters is in between min and max clusters, and if outliers is less than .2
7. If yes, set max purity to purity, bestDBSCAN to that dbscan, bestEpsilon and best minpoints to that value.
8. Return to step 3 and repeat max iterations then Exit the Loop
9. Print the best MinPoints and the Best Epsilon
10. Return bestDBSCAN

Overall this method does not guarantee the very best outcome however it does provide a solution that is at least better than maxIterations other solutions. To find the Min points and Epsilon ranges for Subtask C and E I initially set the Min Points Range and Epsilon Range to 1-25 and then ran the Random search 10 times. I then changed the ranges to the Ranges of the ten solutions found by the algorithm. Then I ran the algorithm 5 more times and set the final ranges to the ranges provided by the 5 solutions.

Subtask D





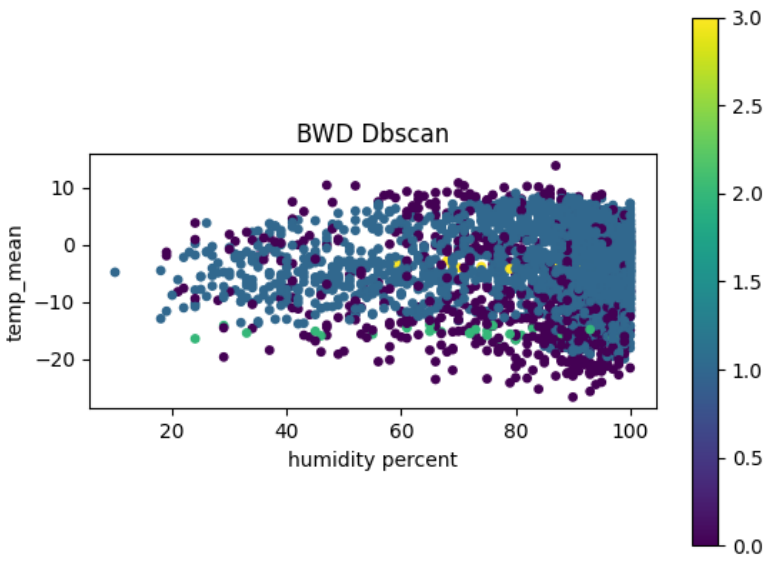


Purity k=3: 0.4; Purity k=5: 0.57
SSE k=3: 186578.5195; SSE k=5: 117025.29877549176
k=3 Cluster Centroid:
[[5.4244, 1.1563, 0.6186, 3.7405, -14.3249, -16.8080, -11.8041, 0.8588] [5.2032, 2.1423, 0.4176, 6.6075, 2.7905, 0.5544, 4.9963, 0.8497] [5.6597, 1.6143, 0.6018, 4.0972, -5.5063, -7.6609, -3.3090, 0.8548]]

k=5 Cluster Centroid:
[[7.0631, 1.5972, 0.689, 2.037, -0.4173, -2.259, 1.4312, 0.948] [6.8679, 1.250, 0.8820, 1.604, -8.094, -10.366, -5.774, 0.925] [4.337, 2.488, 0.3232, 8.9737, 4.558, 2.0454, 7.019, 0.8037] [5.195, 1.165, 0.558, 4.058, -15.632, -18.108, -13.123, 0.852] [2.753, 2.0797, 0.092, 9.635, -4.936, -7.250, -2.578, 0.684]]

Generally for this section the K means model really struggled under the weight of the curse of dimensionality. That can best be seen in the absurdly large SSE values of 186,000 for k=3 and 117025 for k=5. So the error for the data was enormous which would make sense for ten dimensional data. Other than that both K means models had moderate success in predicting the cluster but the k=5 was better at .57 Purity to k=3 Purity of .4

Subtask E



MinPoints Range : [17, 21]
Epsilon Range : [2.18, 2.31]
Best MinPts: 20
Best Epsilon: 2.268073014834371
Number of Classes: 3
BWD Dbscan Purity with Outliers: (0.41, 0.19)

Overall the DBSCAN generated by the process described in the Random Search Explained Section above is not very accurate and has a moderate amount of outliers. The DBSCAN is probably not the best algorithm to describe the data into clusters because of the curse of dimensionality. For my data there are three classes and the outliers. When compared to the BWD Data graph above the BWD Dbscan graph can be seen as quite inaccurate.

Bibliography

OpenAI. (2023, Nov. 6). *ChatGPT* (Nov. 6 version) [Large language model]. https://chat.openai.com/chat

Kevin Arvai. (2023, Nov. 2) *K Means Clustering in Python*. Real Python. https://realpython.com/k-means-clustering-python/

Stanley Juma. (2023, Nov. 2) DBSCAN Algorithm Clustering in Python. https://www.section.io/engineering-education/dbscan-clustering-in-python/

SciKit Learn. (2023, Nov. 2) *sklearn.Cluster.DBSCAN.* https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

W3Schools.com. (2023, Nov 2) *Machine Learning - Confusion Matrix*. https://www.w3schools.com/python/python_ml_confusion_matrix.asp

Appendix 1

To run the program ensure that the pandas, numpy, matplotlib.pyplot, and sklearn.cluster libraries are installed and usable. From there in any IDE run the Python file and all of the desired calculations will be printed into the console and the graphs will appear one by one.

Appendix 2

All of the code I developed for task 3 is below:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, DBSCAN

# SUBTASK A:
# create a purity function
def Purity(a, b, areOutliersPresent):
    confusionMat = np.array([[sum((a == i) & (b == j)) for j in
np.unique(b)] for i in np.unique(a)])
    totalElements = np.sum(confusionMat)
    numOutliers = 0

    # if outliers are present then count them
    if areOutliersPresent:
        for num in a:
            if num == 0:
                numOutliers +=1
```

```
    totalCounts = np.sum(np.max(confusionMat, axis=1))
    purity = totalCounts / totalElements

    if areOutliersPresent:
        outlierPercent = numOutliers / totalElements
        return round(purity, 2), round(outlierPercent, 2)
    else:
        return round(purity, 2)

def CreateScatterPlot(x, y, classifcation, labels):
    title = labels[0]
    xtitle = labels[1]
    ytitle=labels[2]
    plt.figure()
    scatterPlot = plt.scatter(x, y, c=classifcation, s=15)
    plt.colorbar(scatterPlot)
    plt.axis('equal')
    plt.title(title)
    plt.xlabel(xtitle)
    plt.ylabel(ytitle)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()

def RandomSearch(data, groundTruth, minPtsRange, epsilonRange,
maxIterations):
    minClusters = 2
    maxClusters = 17
    maxPurity = 0.0
    bestDBSCAN = None
    bestEpsilon = 0.0
    bestMinPoints = 0

    for i in range(maxIterations):
        iMinPoints = np.random.randint(minPtsRange[0], minPtsRange[1])
        iEpsilon = np.random.uniform(epsilonRange[0], epsilonRange[1])
        iDBSCAN = DBSCAN(min_samples=iMinPoints, eps=iEpsilon)

        iPrediction = iDBSCAN.fit_predict(data) + 1
        iPurity = Purity(iPrediction, groundTruth, True)
        iOutlier = iPurity[1]
```

```
        iClusters = len(set(iPrediction))-1

        if iPurity[0] > maxPurity and minClusters <= iClusters <=
maxClusters and iOutlier < 0.2 :
            maxPurity = iPurity[0]
            bestDBSCAN = iDBSCAN
            bestEpsilon = iEpsilon
            bestMinPoints = iMinPoints

    print("Best MinPts: " + str(bestMinPoints) + "\n")
    print("Best Epsilon: " + str(bestEpsilon) + "\n")
    return bestDBSCAN


def main():
    # Step 0: here we are setting up the data and printing a scatter plot
so I can see the data
    print(" Subtask A \n")
    weatherData = pd.read_csv("Basel_Weather.csv")
    complex8Data = pd.read_csv("Complex8.csv", names=['X', 'Y',
'Cluster'])

    # I need to separate the data so its usable in the plot function
    complex8X = complex8Data['X'].values
    complex8Y = complex8Data['Y'].values
    complex8Cluster = complex8Data['Cluster'].values
    labels = ['Complex8 Data', 'a', 'b']
    CreateScatterPlot(complex8X, complex8Y, complex8Cluster, labels)

    humidityClass = weatherData["humidity_class"].values
    desiredWeatherCols = ["cloud_cover", "global_radiation",
"precipitation", "sunshine", "temp_mean", "temp_min", "temp_max",
"humidity"]
    adjustedWeatherData = weatherData[desiredWeatherCols]

    # changing humidity class from strings to corresponding integers
    for index in range(len(humidityClass)):
        if humidityClass[index] == "Low":
            humidityClass[index] = 1
```

```
        elif humidityClass[index] =="Mid":
            humidityClass[index] = 2
        elif humidityClass[index] == "High":
            humidityClass[index] = 3
        else:
            humidityClass[index] = 0

    # here we are calling and printing the purity function
    print("Given Purity with Outliers: " + str(Purity(complex8Cluster,
complex8Cluster, True)) + "\n")
    print("Given Purity without Outliers: " + str(Purity(complex8Cluster,
complex8Cluster, False)) + "\n")

    # SUBTASK B:
    print("\n Subtask B \n")

    # set up and run kmeans from the scikit-learn lib
    dataColumns = ['X', 'Y']
    kMeansData = complex8Data[dataColumns]

    # creating 4 kmeans object, fitting our data, then storing it
    kMeans8First = KMeans(n_clusters=8, init="random", n_init=10,
max_iter=300, random_state=42)
    kMeans8Second = KMeans(n_clusters=8, init="random", n_init=10,
max_iter=300, random_state=99)
    kMeans12First = KMeans(n_clusters=12, init="random", n_init=10,
max_iter=300, random_state=42)
    kMeans12Second = KMeans(n_clusters=12, init="random", n_init=10,
max_iter=300, random_state=99)

    kMeans8First.fit(kMeansData)
    kMeans8Second.fit(kMeansData)
    kMeans12First.fit(kMeansData)
    kMeans12Second.fit(kMeansData)

    predicted8First = kMeans8First.labels_
    predicted8Second = kMeans8Second.labels_
    predicted12First = kMeans12First.labels_
    predicted12Second = kMeans12Second.labels_
```

```
    print8First = "K=8 First Round \nPurity with outliers: " +
str(Purity(predicted8First, complex8Cluster, True)) +  ". Purity without
Outliers: " + str(Purity(predicted12First, complex8Cluster, False)) + "\n"
    print8Second = "K=8 Second Round \nPurity with outliers: " +
str(Purity(predicted8Second, complex8Cluster, True)) +  ". Purity without
Outliers: " + str(Purity(predicted8Second, complex8Cluster, False)) + "\n"
    print12First = "K=12 First Round \nPurity with outliers: " +
str(Purity(predicted12First, complex8Cluster, True)) +  ". Purity without
Outliers: " + str(Purity(predicted12First, complex8Cluster, False)) + "\n"
    print12Second = "K=12 Second Round \nPurity with outliers: " +
str(Purity(predicted12Second, complex8Cluster, True)) +  ". Purity without
Outliers: " + str(Purity(predicted12Second, complex8Cluster, False)) +
"\n"

    print(print8First)
    print(print8Second)
    print(print12First)
    print(print12Second)

    # put those plots up
    labels8First = ["K=8 First Iteration", "a", "b"]
    labess8Second = ["K=8 Second Iteration", "a", "b"]
    labels12First = ["K=12 First Iteration", "a", "b"]
    labels12Second = ["K=12 Second Iteration", "a", "b"]

    CreateScatterPlot(complex8X, complex8Y, predicted8First, labels8First)
    CreateScatterPlot(complex8X, complex8Y, predicted8Second,
labess8Second)
    CreateScatterPlot(complex8X, complex8Y, predicted12First,
labels12First)
    CreateScatterPlot(complex8X, complex8Y, predicted12Second,
labels12Second)

    # SUBTASK C: Develop the search procedure
    print("\n Subtask C \n")
    task3Data = np.hstack((complex8X.reshape(-1, 1), complex8Y.reshape(-1,
1)))

    minPointsRange = [2,4]
    epsilonRange = [13.6,15.0]
```

```
    print("MinPoints Range : " + str(minPointsRange) + "\n")
    print("Epsilon Range: " + str(epsilonRange) + "\n")

    bestComplexDBSCAN = RandomSearch(task3Data, complex8Cluster,
minPointsRange, epsilonRange, 500)
    bestClassifications = bestComplexDBSCAN.fit_predict(task3Data)
    bestClassesOutlierAdjusted = bestClassifications + 1
    print("Number of Classes: " + str(max(bestClassesOutlierAdjusted)) +
"\n")


    print("DBSCAN Purity with Outliers: " +
str(Purity(bestClassesOutlierAdjusted, complex8Cluster, True)) + "\n")
    labelsComplex8Dbscan = ["DBSCAN", "a", "b"]
    CreateScatterPlot(complex8X, complex8Y, bestClassesOutlierAdjusted,
labelsComplex8Dbscan)

    # SUBTASK D: Run Kmeans k=3 + k=5 for BWD
    print("\n Subtask D \n")

    # bwd  k=3 setup
    bwdK3 = KMeans(n_clusters=3, init="random", n_init=10,  max_iter=300,
random_state=42)
    bwdK3.fit(adjustedWeatherData)
    predictedbwdK3 = bwdK3.labels_
    predictedbwdK3OutlierAdj = predictedbwdK3 + 1

    # bwd k=5 setup
    bwdK5 = KMeans(n_clusters=5, init="random", n_init=10,  max_iter=300,
random_state=42)
    bwdK5.fit(adjustedWeatherData)
    predictedbwdK5 = bwdK5.labels_
    predictedbwdK5OutlierAdj = predictedbwdK5 + 1

    # I had trouble with plotting humidty so multiplying by 100 turns
humidity into a percent and better for scaling the graph
    adjustedHumidityValues = adjustedWeatherData["humidity"].values *
100.0
```

```
    labelsBwdData = ["BWD Data", "humidity percent", "temp_mean"]
    labelsBwdk3 = ["BWD K=3", "humidity percent", "temp_mean"]
    labelsBwdk5 = ["BWD K=5", "humidity percent", "temp_mean"]

    CreateScatterPlot(adjustedHumidityValues,
adjustedWeatherData["temp_mean"].values, humidityClass, labelsBwdData)
    CreateScatterPlot(adjustedHumidityValues,
adjustedWeatherData["temp_mean"].values, predictedbwdK3OutlierAdj,
labelsBwdk3)
    CreateScatterPlot(adjustedHumidityValues,
adjustedWeatherData["temp_mean"].values, predictedbwdK5OutlierAdj,
labelsBwdk5)

    # purity calculations
    print("k=3 Purity: " + str(Purity(predictedbwdK3OutlierAdj,
humidityClass, False)) + "\n")
    print("k=5 Purity: " + str(Purity(predictedbwdK5OutlierAdj,
humidityClass, False)) + "\n")

    # cluster centroids
    centroidBWDK3 = bwdK3.cluster_centers_
    centroidBWDK5 = bwdK5.cluster_centers_

    print("k=3 Cluster Centroid:\n" + str(centroidBWDK3) + "\n")
    print("k=5 Cluster Centroid:\n" + str(centroidBWDK5) + "\n")

    # SSE
    sseBWDK3 = bwdK3.inertia_
    sseBWDK5 = bwdK5.inertia_

    print("k=3 SSE: " + str(sseBWDK3) + "\n")
    print("k=5 SSE: " + str(sseBWDK5) + "\n")

    # SUBTASK E
    print("\n Subtask E\n")
    minPointsRange = [17,21]
    epsilonRange = [2.18,2.31]

    print("MinPoints Range : " + str(minPointsRange) + "\n")
    print("Epsilon Range : " + str(epsilonRange) + "\n")
```

```
    bestBwdDBSCAN = RandomSearch(adjustedWeatherData, humidityClass,
minPointsRange, epsilonRange, 500)
    bestBwdClassifications =
bestBwdDBSCAN.fit_predict(adjustedWeatherData)
    bestBwdClassesOutlierAdjusted = bestBwdClassifications + 1

    print("Number of Classes: " + str(max(bestBwdClassesOutlierAdjusted))
+ "\n")

    labelsBwdDbscan = ["BWD Dbscan", "humidity percent", "temp_mean"]
    CreateScatterPlot(adjustedHumidityValues,
adjustedWeatherData["temp_mean"].values, bestBwdClassesOutlierAdjusted,
labelsBwdDbscan)

    print("BWD Dbscan Purity with Outliers: " +
str(Purity(bestBwdClassesOutlierAdjusted, humidityClass, True)) + "\n")

main()
```