1) Machine learning definitions:

What types of Machine Learning, if any, best describe the following scenarios? Please provide a brief explanation. Ex: supervised, unsupervised, reinforcement learning, and no learning.

(i) The exact specifications of each coin are measured by an engineer. The vending machine recognizes a given coin based on these specifications.

Supervised learning, granted it is categorical, the vending machine is given a set of specifications which will allow it to place a specific coin in a category. Since it is observed and directed, it is supervised.

(ii) An algorithm is presented with a large set of labeled coins and uses this data to infer decision boundaries, based on which the vending machine classifies new coins.

Unsupervised learning, the model worked on its own to discover information that might not be clear at first for a human. I will draw conclusion based on unlabeled data, making it unsupervised learning.

(iii) An algorithm is successively presented with coins. Each time the algorithm makes a decision about the coin type and checks the correctness of the decision with the engineer. Based on the engineer's answer, the algorithm refines the process with which it makes the decision for the next coin.

Reinforcement learning. Since it is being rewarded/punished for its decisions, the model will be reinforced consistently allowing it to become a stronger model with time. This is the premise of reinforcement learning, to strengthen its decision-making skills through new generations.

2) Classifying benign vs malignant tumors:

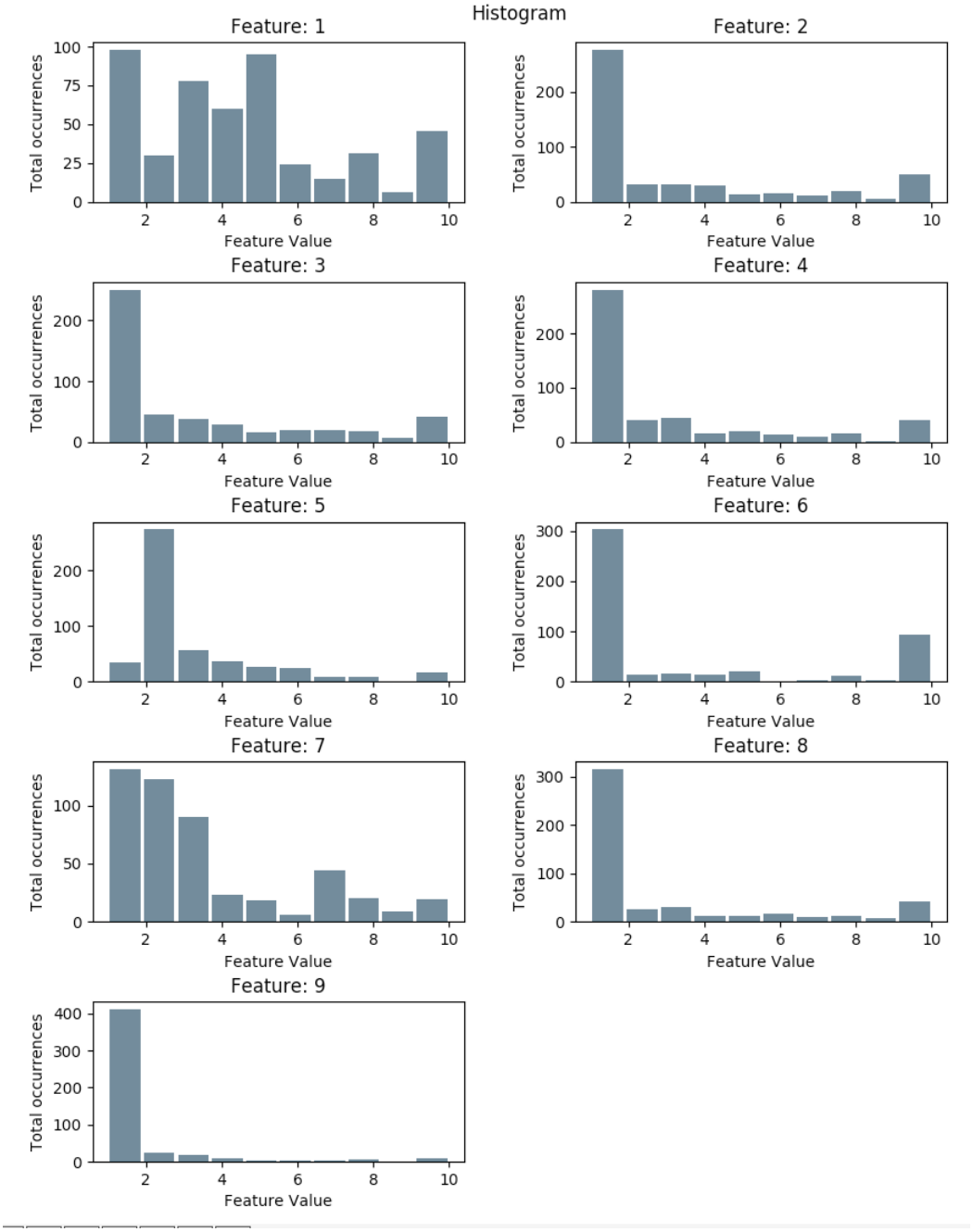
(a) Data exploration:

(i)

	Benign	Malignant
Totals:	329	154

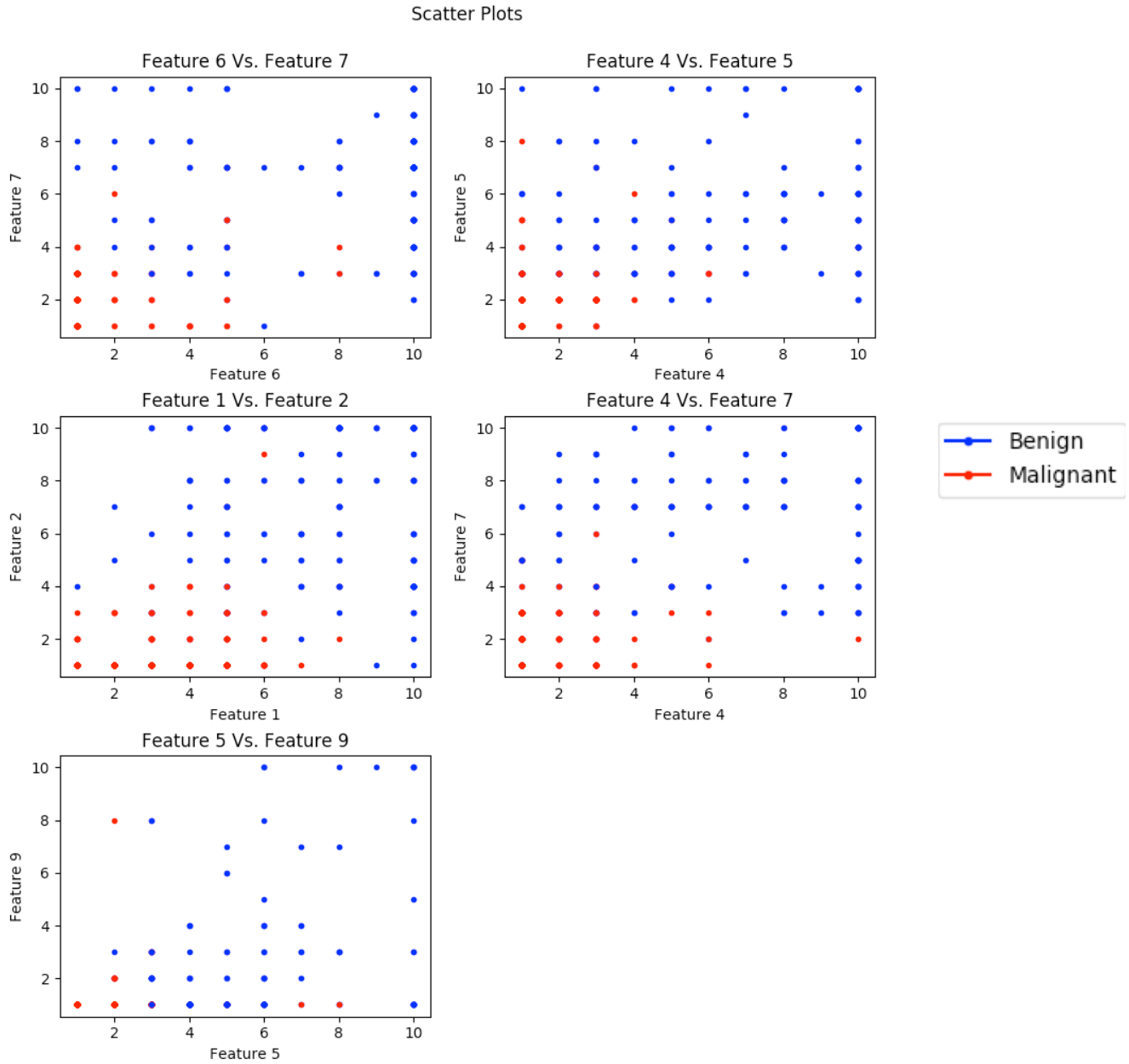
Observing the training data, we see that Benign (class A) has a lot more samples. This gives the implication that a Malignant sample will most likely be misclassifies as compared to the Benign since our model can have a deeper understanding of the features of Benign. Our samples are not equally distrusted since almost seventy percent of our testing samples is from class A.

(ii)



The features are distributed non-uniformly, where we can see that there are many features with a value of between 1 – 3. The distribution differs between the features; however, the general behavior can be observed. They tend to represent a right skewed graph.

(iii)

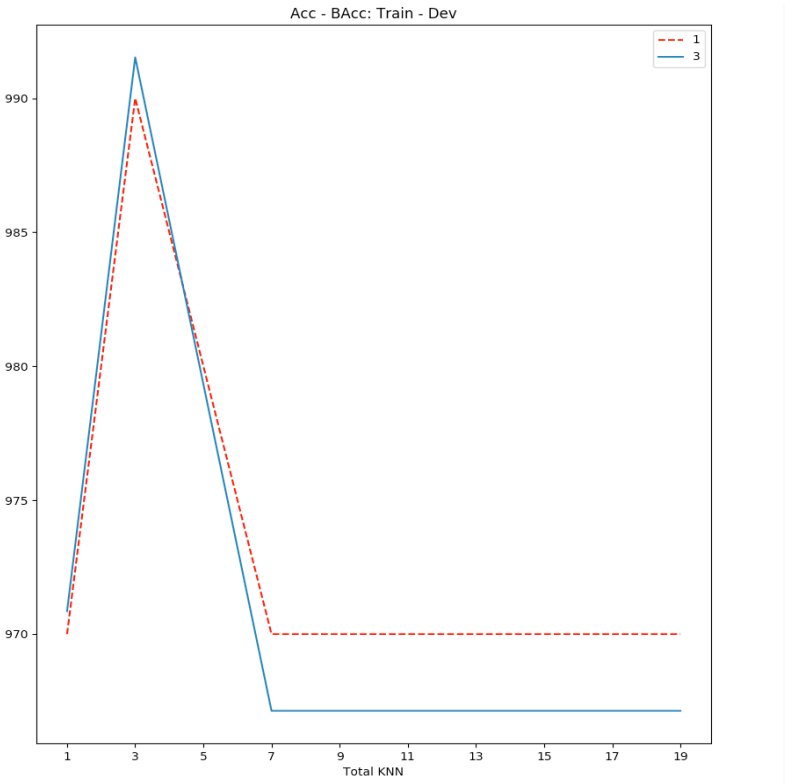


Observing the scatter plots, Benign shows to have a higher frequency of higher values. One can conclude, based off of the graphs, that if you have a high bare nucleus or cell size, they are most likely going to be categorized as Benign. The data seems to be clustered enough to group but not separate enough to be prone to misclassifications.

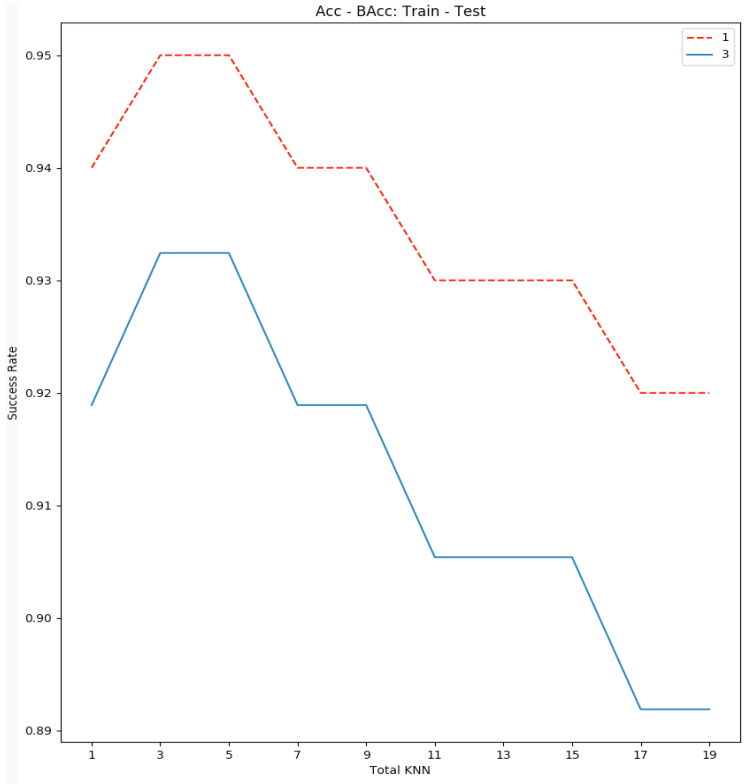
(b)

(i) Implemented by creating a KNN and Euclidean function. (on Code sheet)

(ii)



	1	3	5	7	9	11	13	15	17	19
Acc	0.97	0.99	0.98	0.97	0.97	0.97	0.97	0.97	0.97	0.97
BAcc	0.971	0.992	0.979	0.967	0.967	0.967	0.967	0.967	0.967	0.967



	1	3	5	7	9	11	13	15	17	
Acc	0.94	0.95	0.95	0.94	0.94	0.93	0.93	0.93	0.92	
BAcc	0.919	0.932	0.932	0.919	0.919	0.905	0.905	0.905	0.892	

Observing both, the training vs dev and training vs test, we can see that, based on my methods of implementation, a nearest neighbor of **three** is an ideal choice to represent the hyper-parameter of k. One can observe that as k's increase, the probability of a successful predication drops. This can be inferred by noting that the samples are not too spread out and there is an unbalanced number of samples in each class. This can result in more Malignant to be represented as Benign since it has a higher sample population.

(iii)

	Class A (Dev)	Class A(Test)	Class B(Dev)	Class B(Test)	Acc	BAcc
K_1=3	59	58	41	41	0.99	0.99152
K_2=5	59	58	41	40	0.98	0.97933

	Class A (Train)	Class A(Test)	Class B(Train)	Class B(Test)	Acc	BAcc
K_1=3	63	63	37	32	0.95	0.932432
K_2=5	63	63	37	32	0.95	0.932432

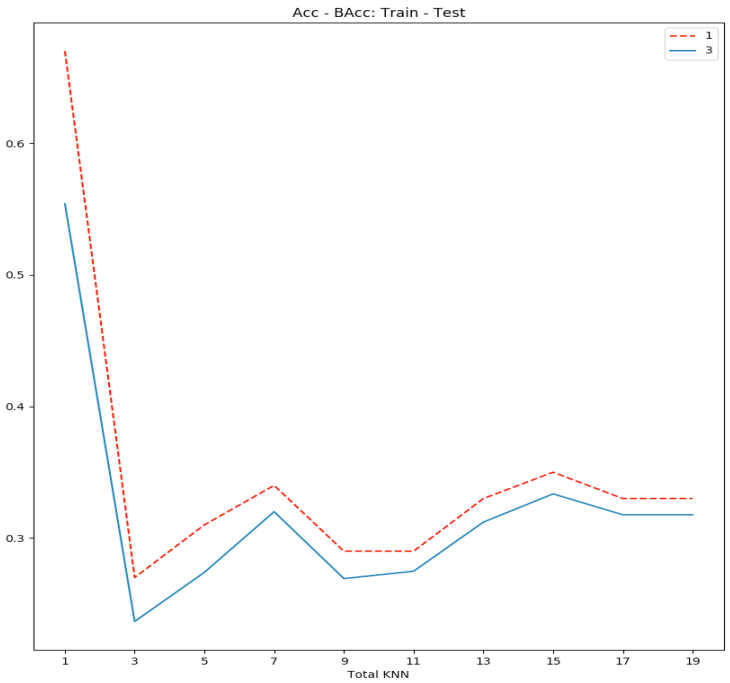
You can observe that the training data, along with the development data, allowed for a strong success rate of the testing data. Although not as high of a success rate as the dev data, the test data was about 93 percent success rate.

(iv) **Bonus**: x

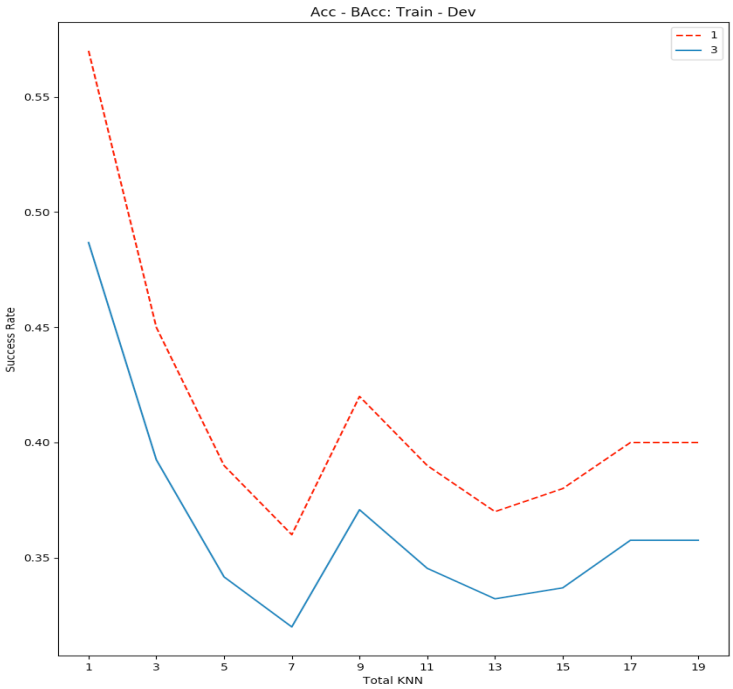
Using a cosine similarity approach towards determine the labels shows to not be an ideal decision making choice. Granted the points aren't spread based on angle with respects to the origin, it holds a low success rate.

	Class A (Dev)	Class A(Test)	Class B(Dev)	Class B(Test)	Acc	BAcc
K_1=3	59	42	41	3	0.45	0.3925
K_2=5	59	36	41	3	0.39	0.3416

	Class A (Train)	Class A(Test)	Class B(Train)	Class B(Test)	Acc	BAcc
K_1=3	63	23	37	4	0.27	0.23659
K_2=5	63	26	37	5	0.31	0.27391



	1	3	5	7	9	11	13	15	17	19
Acc	0.67	0.27	0.31	0.34	0.29	0.29	0.33	0.35	0.33	0.33
BAcc	0.554	0.237	0.274	0.320	0.269	0.275	0.312	0.334	0.318	0.318



	1	3	5	7	9	11	13	15	17	
Acc	0.57	0.45	0.39	0.36	0.42	0.39	0.37	0.38	0.4	
BAcc	0.487	0.393	0.342	0.320	0.371	0.345	0.332	0.337	0.358	

CODE

```
import matplotlib.pyplot as plt
import numpy as np
import random
import pandas as pd
import math


def ACC(train, test):
    acc = []
    bacc = []
    classesCount = totalInClasses(test)
    for ks in range(1, 20, 2):  # for each k
        predictions = []
        correctA = 0
        correctB = 0
        correctSamp = 0
        for i in range(len(test[0])):  # for each test sample
            testSample = []
            for j in range(len(test) - 1):
                testSample.append(test[j][i])
            knn = KNN(train, testSample, ks)  # returns list of k values:
[(euclidean, class)]
            mostCommon = max(set(knn), key=knn.count)  # most common class
            predictions.append(mostCommon)
            if mostCommon == 2.0 and test[9][i] == 2.0:
                correctA += 1
            if mostCommon == 4.0 and test[9][i] == 4.0:
                correctB += 1
            if predictions[i] == test[9][i]:
                correctSamp += 1
        acc.append(correctSamp / len(test[9]))
        corrBalanced = 0.5 * ((correctA / classesCount[0]) + (correctB /
classesCount[1]))
        bacc.append(corrBalanced)
    return acc, bacc


def graphACC(a, b, titles):
    acc = (ACC(a, b))

    plt.figure()
    plt.title('Acc - BAcc: ' + titles[0] + ' - ' + titles[1])
    plt.xlabel("Total KNN")
    plt.ylabel("Success Rate")
    plt.subplots_adjust(bottom=0.15, left=0.1, top=0.95)
    x = [i for i in np.arange(1, 20, 2)]
    plt.xticks(np.arange(1, 21, step=2))
    plt.plot(x, acc[0], 'r--', x, acc[1])

    for i in range(len(acc[1])):
        acc[1][i] = '%.3f' % acc[1][i]
    d = {'1': [acc[0][0], acc[1][0]], '3': [acc[0][1], acc[1][1]],
         '5': [acc[0][2], acc[1][2]], '7': [acc[0][3], acc[1][3]],
```

```
         '9': [acc[0][4], acc[1][4]], '11': [acc[0][5], acc[1][5]],
         '13': [acc[0][6], acc[1][6]], '15': [acc[0][7], acc[1][7]],
         '17': [acc[0][8], acc[1][8]], '19': [acc[0][9], acc[1][9]]}

    mylabels = ["Acc", "BAcc"]
    makeTable(d, mylabels)


def graphNFeatures(train, n):
    plt.figure()
    plt.suptitle('Histogram')
    plt.subplots_adjust(bottom=0.15, left=0.1, hspace=0.5, wspace=0.30,
top=0.95)  # or whatever
    for j in range(n):
        plt.subplot(5, 2, j + 1)
        plt.xlabel("Feature Value")
        plt.ylabel("Total occurrences")
        plt.title("Feature: " + str(j + 1))
        for i in range(len(train)):
            plt.hist(train[j], bins=len(train), rwidth=0.9, color='#607c8e')


def totalInClasses(train):
    class1 = 0
    class2 = 0
    for i in range(len(train[0])):
        if int(train[9][i]) == 2:
            class1 += 1
        else:
            class2 += 1
    return [class1, class2]


def plotNFeatures(train, n):
    plt.figure()
    plt.suptitle('Scatter Plots')
    plt.subplots_adjust(bottom=0.12, left=0.1, hspace=0.30, top=0.92)  # or
whatever
    plt.xlim(0, 11)
    plt.ylim(0, 11)
    plt.xticks(range(0, 11))
    plt.yticks(range(0, 11))

    for j in range(n):
        plt.subplot(math.ceil(n / 2), math.floor(n / 2), j + 1)
        ran = random.sample(range(len(train) - 1), 2)
        feature1 = ran[0]
        feature2 = ran[1]
        plt.xlabel("Feature " + str(feature1 + 1))
        plt.ylabel("Feature " + str(feature2 + 1))
        plt.title("Feature " + str(feature1 + 1) + " Vs. Feature " +
str(feature2 + 1))
        for i in range(len(train[0])):
            if int(train[9][i]) == 2:
                plt.plot(int(train[feature1][i]), int(train[feature2][i]),
```

```
marker='o', markersize=3, color="red")
            else:
                plt.plot(int(train[feature1][i]), int(train[feature2][i]),
marker='o', markersize=3, color="blue")


def makeTable(table, mylabel):
    dcsummary = pd.DataFrame(table, index=mylabel)
    plt.legend(dcsummary.columns, prop={'size': 10})
    plt.table(cellText=dcsummary.values,
              colWidths=[0.25] * len(dcsummary.columns),
              rowLabels=dcsummary.index,
              colLabels=dcsummary.columns,
              cellLoc='center', rowLoc='center',
              loc='bottom', bbox=[0, -0.06 * len(mylabel) - 0.02,
                                  0.1 * len(table) + 0.1, 0.01 * len(mylabel)
+ 0.06])  # x, y, length, height


def euclidean(train, test):
    out = 0
    for i in range(len(train)):
        out += (train[i] - test[i]) ** 2
    return math.sqrt(out)


def cosSim(train, test):
    dot_product = np.dot(train, test)
    norm_a = np.linalg.norm(train)
    norm_b = np.linalg.norm(test)
    return dot_product / (norm_a * norm_b)


def KNN(train, test, k):
    knn = []
    tumor = []
    for i in range(len(train[0])):
        trainSample = []
        for j in range(len(train) - 1):
            trainSample.append(train[j][i])
        knn.append((cosSim(trainSample, test), train[9][i]))

    for i in range(k):
        smallestDist = min(knn, key=lambda x: x[0])
        tumor.append(smallestDist)
        knn.remove(smallestDist)

    return [w[1] for w in tumor]  # gets only the classes


def main():
    trainingData = 'hw1_question1_train.csv'
    testingData = 'hw1_question2_test.csv'
    devData = 'hw1_question2_dev.csv'
```

```
    train = np.loadtxt(trainingData, delimiter=",", unpack=True)
    test = np.loadtxt(testingData, delimiter=",", unpack=True)
    dev = np.loadtxt(devData, delimiter=",", unpack=True)

    graphACC(train, dev, ["Train", "Dev"])
    graphACC(train, test, ["Train", "Test"])

    # ----------------------  HISTOGRAM ------------------------
    graphNFeatures(train, 9)

    # ----------------------  PLOT ------------------------
    plotNFeatures(train, 5)
    table = totalInClasses(train)
    d = {'Benign': [table[0]], 'Malignant': [table[1]]}
    makeTable(d, ["Totals:"])

    # ----------------------  SHOW ------------------------
    plt.show()


if __name__ == "__main__":
    main()
```