In [1]:

```
# import libraries
import numpy as np
import pandas as pd
```

Question 3:

a)

In [2]:

```
file = pd.read_csv('wine-training')
file1 = pd.read_csv('wine-test')
file.to_csv('wine-training.csv')
file1.to_csv('wine-test.csv')
```

In [3]:

```
wineTrain_DF = pd.read_csv('wine-training.csv')
wineTest_DF = pd.read_csv('wine-test.csv')
```

In [4]:

```
result = pd.DataFrame('Alcohol Malic_acid Ash Alcalinity_of_ash Magnesium Total_phen
result2 = pd.DataFrame('Alcohol Malic_acid Ash Alcalinity_of_ash Magnesium Total_phe
for i in range(len(wineTest_DF)):
    data = pd.DataFrame(wineTest_DF['Alcohol Malic_acid Ash Alcalinity_of_ash Magnes
    result = pd.concat([result, data], axis=1)
for i in range(len(wineTest_DF)):
    data = pd.DataFrame(wineTrain_DF['Alcohol Malic_acid Ash Alcalinity_of_ash Magne
    result2 = pd.concat([result2, data], axis=1)
```

In [5]:

```
wineTestData = result.transpose()
wineTestData.columns = wineTestData.iloc[0]
wineTestData = wineTestData.iloc[1:]
for i in wineTestData.columns:
    wineTestData[i] = pd.to_numeric(wineTestData[i])
wineTestData
```

Out[5]:

	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonfla
0	12.70	3.55	2.36	21.5	106.0	1.70	1.20	
0	12.20	3.03	2.32	19.0	96.0	1.25	0.49	
0	14.13	4.10	2.74	24.5	96.0	2.05	0.76	
0	13.05	1.65	2.55	18.0	98.0	2.45	2.43	
0	14.19	1.59	2.48	16.5	108.0	3.30	3.93	
...	...	...	...	...	...	...	...	
0	11.84	0.89	2.58	18.0	94.0	2.20	2.21	
0	13.73	1.50	2.70	22.5	101.0	3.00	3.25	
0	13.39	1.77	2.62	16.1	93.0	2.85	2.94	
0	11.62	1.99	2.28	18.0	98.0	3.02	2.26	
0	13.90	1.68	2.12	16.0	101.0	3.10	3.39	

89 rows × 14 columns

In [6]:

```
wineTrainData = result2.transpose()
wineTrainData.columns = wineTrainData.iloc[0]
wineTrainData = wineTrainData.iloc[1:]
for i in wineTrainData.columns:
    wineTrainData[i] = pd.to_numeric(wineTrainData[i])
wineTrainData
```

Out[6]:

	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonfla
0	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
0	12.25	1.73	2.12	19.0	80.0	1.65	2.03	
0	11.82	1.47	1.99	20.8	86.0	1.98	1.60	
0	13.05	2.05	3.22	25.0	124.0	2.63	2.68	
0	13.29	1.97	2.68	16.8	102.0	3.00	3.23	
...	...	...	...	...	...	...	...	
0	12.77	2.39	2.28	19.5	86.0	1.39	0.51	
0	12.79	2.67	2.48	22.0	112.0	1.48	1.36	
0	13.28	1.64	2.84	15.5	110.0	2.60	2.68	
0	12.96	3.45	2.35	18.5	106.0	1.39	0.70	
0	12.47	1.52	2.20	19.0	162.0	2.50	2.27	

89 rows × 14 columns

In [7]:

```
featureList = ['Alcohol','Malic_acid','Ash','Alcalinity_of_ash','Magnesium','Total_pl
               'Flavanoids','Nonflavanoid_phenols','Proanthocyanins','Color_intensity
gigaset = pd.DataFrame(np.concatenate((wineTrainData, wineTestData), axis=0))
columnList = ['Alcohol','Malic_acid','Ash','Alcalinity_of_ash','Magnesium','Total_ph
              'Flavanoids','Nonflavanoid_phenols','Proanthocyanins','Color_intensity
gigaset.columns = columnList
gigaset = gigaset.sample(frac=1)
```

In [115]:

```
def accuracyScore(predicted_vals, y):
    score=0
    for i in range(len(y)):
        if y[i] == predicted_vals[i]:
            score = score + 1
    return score/len(y)

class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, train_data_X, train_data_Y):
        # min-max normalisation happens here
        self.train_data_X = (train_data_X - train_data_X.min(axis=0))/(train_data_X.
        self.train_data_Y = train_data_Y

    def predict(self, test_data, features):
        # min-max normalisation happens here
        testData = test_data
        testData[features] = (testData[features] - testData[features].min(axis=0))/(
        predictions = []
        for i in range(len(test_data)):
            distances = []
            for j in range(len(self.train_data_X)):
                x1 = testData[features].iloc[i]
                x2 = self.train_data_X.iloc[j]
                # distance is calculated here
                distance = np.sqrt(np.sum((x1-x2)**2))
                distances.append((distance, self.train_data_Y.iloc[j]))
            distances.sort()
            neighbors = distances[:self.k]
            classes = [neighbor[1] for neighbor in neighbors]
            prediction = max(set(classes), key=classes.count)
            predictions.append(prediction)
        return predictions
```

In [116]:

```
# Make predictions
model = KNN(1)
model2 = KNN(3)
model3 = KNN(89)
model.fit(wineTrainData[featureList], wineTrainData['Class'])
model2.fit(wineTrainData[featureList], wineTrainData['Class'])
model3.fit(wineTrainData[featureList], wineTrainData['Class'])
predictions = model.predict(wineTestData,featureList)
predictions2 = model2.predict(wineTestData,featureList)
predictions3 = model3.predict(wineTestData,featureList)
```

Predicted class labels:

In [124]:

```
predictions
```

```
1,
3,
1,
1,
3,
3,
2,
2,
3,
2,
3,
3,
1,
1,
2,
1,
3,
2,
2,
1,
1,
```

In [117]:

```
classWT = wineTestData['Class'].tolist()
```

In [118]:

```
print("K=1 Accuracy Score: ", accuracyScore(predictions, classWT))
print("K=3 Accuracy Score: ", accuracyScore(predictions2, classWT))
print("K=89 Accuracy Score: ", accuracyScore(predictions3, classWT))
```

```
K=1 Accuracy Score:  0.9438202247191011
K=3 Accuracy Score:  0.9213483146067416
K=89 Accuracy Score:  0.39325842696629215
```

Accuracy Score when k = 1 is: 0.9438202247191011

b)

When k=3, the score is 0.9213483146067416 which is a lower score than when k=1. This is normal because a smaller k value means a smaller number of k-neighbors when fitting the data, which usually results in more overfitting of data. When there's a larger k value like 3 the data gets more underfit, which means that the model trains worse on the training data but will most likely perform better with new data than when k is set to 1.

c)

K-Nearest Neighbor method has a set of advantages and disadvantages like every predictive model. Advantages include its simplicity, where both training and predicting are efficient processes, and the fact that it can easily be adapted for regression rather than classifiying. However KNN struggles to handle higher

dimensional data and nominal features, as well as noisy and missing data.

In general K-Nearest Neighbor method is a machine learning model that is easy to implement and has a low to zero training period as the data itself is a model used to reference future predictions, and distances are calculated using distance formulas; in this case being the Euclidian formula. However a lot of input data preprocessing is often needed for the model to be able to use i, proccesses which include feature scaling and accounting for missing values and noisy data.

d)

Implementing k-folds is a way of verifying and validating a machine learning model's performance through splitting a dataset up into multiple smaller sets named folds.

The number of folds equals the value of K, but the model is trained on K-1 of the folds then tested on the remaining fold, and this process is repeated K number of times. If K = 5, you split the training data into 5 equally sized folds with the data being randomly distributed in order to ensure that theres no biases when training the data multiple times.

The average prediction accuracies of these different training cycles is then calculated to give a more accurate view on how the model will perform with new unseen data, by simply putting all the accuracy scores into an array then dividing the sum of the array with the total number of folds.