In [36]:
```
# from google.colab import drive
import numpy as np

# makes printing more human-friendly
np.set_printoptions(precision=3,suppress=True)
```

In [37]:
```
# a) Load data

colab = False  # Set to True if using colab
if colab:
    # May require changing paths to file
    drive.mount('/content/drive')
    with open('/content/drive/Ex_PC_data.csv', 'r') as f:
      data = np.genfromtxt(f,delimiter=',')
else:
    # May require changing paths to file
    with open('Ex_PC_data.csv', 'r') as f:
      data = np.genfromtxt(f,delimiter=',')

X = data[:,:-1]
y = data[:,-1]
```

In [38]:
```
# b) number of samples, features dimension, the number of classes

num_samples = len(y)
num_feats = len(X[0])
num_classes = len(np.unique(y))
num_samples_per_class = {value: np.sum(y == value) for value in np.unique(y)
print(num_samples_per_class)

print(f'num of samples: {num_samples}')
print(f'num of feature dimensions: {num_feats}')
print(f'num of classes: {num_classes}')
for cls in num_samples_per_class:
  print(f'class {cls} has {num_samples_per_class[cls]} samples')
```

```
{1.0: 59, 2.0: 71, 3.0: 48}
num of samples: 178
num of feature dimensions: 13
num of classes: 3
class 1.0 has 59 samples
class 2.0 has 71 samples
class 3.0 has 48 samples
```

In [39]:
```
# c) check nan, data imputation
from sklearn.impute import KNNImputer

if np.sum(np.isnan(X)):
  print('Total of NaN before imputation:', np.sum(np.isnan(X)))
  X = KNNImputer(n_neighbors=2, weights="uniform").fit_transform(X)
  print('Total of NaN after imputation:', np.sum(np.isnan(X)))
else:
  print('no NaN')
```

```
Total of NaN before imputation: 6
Total of NaN after imputation: 0
```

Q) How are the missing values completed when using KNNImputer?

A) It averages the feature from the nearest n neighbors and weighs them based on the weights variable

In [40]:
```
# d) partition 80/20
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

print('training data size: ', X_train.shape[0])
print('testing data size: ', X_test.shape[0])
```

```
training data size:  142
testing data size:  36
```

In [41]:
```
# e) standardize to -5 to 5

X_train_min = np.min(X_train, axis=0)
X_train_max = np.max(X_train, axis=0)

X_train_standardized = ((X_train - X_train_min) / (X_train_max - X_train_min
print('min training data in each dimension, after standardization:', np.min(
print('max training data in each dimension, after standardization:', np.max(

# Warning: When standardizing the test set, we should use statistics like mi
X_test_standardized = ((X_test - X_train_min) / (X_train_max - X_train_min))
print('min testing data in each dimension, after standardization:', np.min(X
print('max testing data in each dimension, after standardization:', np.max(X
```

```
min training data in each dimension, after standardization: [-5. -5. -5. -5.
-5. -5. -5. -5. -5. -5. -5. -5. -5.]
max training data in each dimension, after standardization: [5. 5. 5. 5. 5.
5. 5. 5. 5. 5. 5. 5. 5.]
min testing data in each dimension, after standardization: [-4.    -5.305
-3.182 -2.732 -4.13  -3.724 -4.662 -4.245 -5.032 -5.409
 -4.268 -4.89  -4.63 ]
max testing data in each dimension, after standardization: [3.605 2.556 3.02
1 2.423 0.761 3.    2.574 4.434 1.551 2.247 1.748 4.341
 6.048]
```

In [42]:
```
# f) standardize to 0 mean, unit variance

X_train_mean = X_train.mean(axis=0)
X_train_std = X_train.std(axis=0)
X_train_standardized = (X_train - X_train_mean) / X_train_std
print('mean training data in each dimension, after standardization:', np.mea
print('std training data in each dimension, after standardization:', np.std(

# Warning: When standardizing the test set, we should use statistics like mi
X_test_standardized = (X_test - X_train_mean) / X_train_std
print('min testing data in each dimension, after standardization:', np.mean(
print('max testing data in each dimension, after standardization:', np.std(X
```

```
mean training data in each dimension, after standardization: [ 0. -0.  0.  0
. -0.  0.  0. -0.  0.  0.  0.  0. -0.]
std training data in each dimension, after standardization: [1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1.]
min testing data in each dimension, after standardization: [ 0.133 -0.161  0
.101  0.033 -0.238  0.047  0.127 -0.232 -0.153  0.001
  0.023  0.13   0.197]
max testing data in each dimension, after standardization: [0.948 0.87  0.91
0.825 0.846 0.907 0.977 0.823 0.9   0.977 0.881 0.91
 1.182]
```

In [43]:

```
# g) k fold
from sklearn.model_selection import KFold

kf = KFold(n_splits=3)

# Count the class distributions in each partition
for train_index, val_index in kf.split(X_train):
  print('training class distribution: ', np.count_nonzero(y_train[train_inde
  print('validation class distribution: ', np.count_nonzero(y_train[val_inde
```

```
training class distribution:  94
validation class distribution:  48
training class distribution:  95
validation class distribution:  47
training class distribution:  95
validation class distribution:  47
```

In [44]:

```
if colab:
    drive.flush_and_unmount()
```