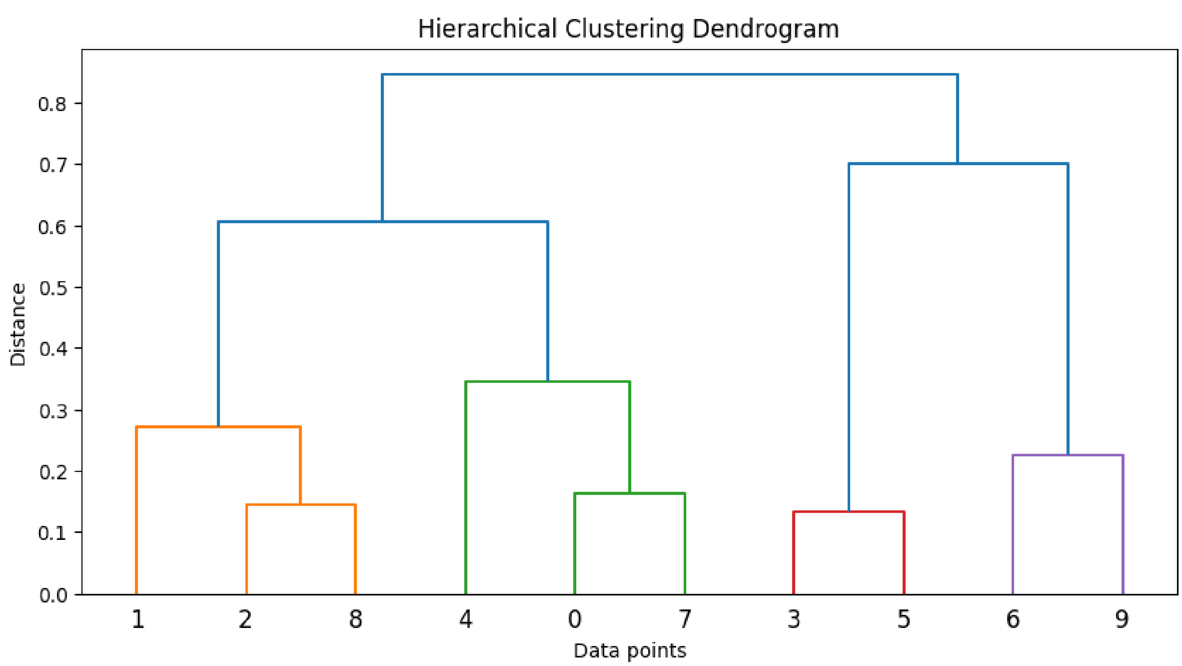
Machine Learning LAB - Assessment 4

Jeevan Yohan Varghese - 20BCE0083

1. Implement Hierarchical clustering.

In [32]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial.distance import pdist
```

In [33]:
```
X = np.random.rand(10, 2)

# Distance matrix
dist_matrix = pdist(X)

# Perform hierarchical clustering
linkage_matrix = linkage(dist_matrix, method='complete')

# Dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix, color_threshold=0.5*max(linkage_matrix[:, 2]))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data points')
plt.ylabel('Distance')
plt.show()
```



2. Implement Gaussian Mixture Model Using the Expectation Maximization

```
In [34]: from sklearn.mixture import GaussianMixture
import numpy as np

np.random.seed(42)
x = np.concatenate([np.random.normal(0, 1, 500), np.random.normal(5, 1, 500)

# GMM with 2 components
gmm = GaussianMixture(n_components=2, random_state=42)

# Training
gmm.fit(x)

# print the means and standard deviations of the two Gaussian components
print("Means:", gmm.means_.flatten())
print("Standard Deviations:", np.sqrt(gmm.covariances_).flatten())
```

```
Means: [-4.06425046e-03  5.02442983e+00]
Standard Deviations: [0.96338193 0.98477776]
```

3. Evaluating ML algorithm with balanced and unbalanced datasets.

```
In [23]: import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
```

```
In [26]: #balanced dataset
X_bal, y_bal = make_classification(n_samples=1000, n_features=10, n_classes=

#unbalanced dataset
X_unbal, y_unbal = make_classification(n_samples=1000, n_features=10, n_clas


X_train_bal, X_test_bal, y_train_bal, y_test_bal = train_test_split(X_bal, y
X_train_unbal, X_test_unbal, y_train_unbal, y_test_unbal = train_test_split(

# Training on balanced dataset
model_balanced = LogisticRegression(random_state=123)
model_balanced.fit(X_train_bal, y_train_bal)

# Testing on balanced dataset
y_pred_bal = model_balanced.predict(X_test_bal)

# Balanced model - evaluation
acc_bal = accuracy_score(y_test_bal, y_pred_bal)
print("Accuracy on balanced dataset: {:.2f}%".format(acc_bal * 100))
print(classification_report(y_test_bal, y_pred_bal))
```

```
Accuracy on balanced dataset: 95.50%
              precision    recall  f1-score   support

           0       0.94      0.97      0.96        99
           1       0.97      0.94      0.95       101

    accuracy                           0.95       200
   macro avg       0.96      0.96      0.95       200
weighted avg       0.96      0.95      0.95       200
```

In [30]:

```python
# training on unbalanced data
model_unbalanced = LogisticRegression(random_state=123)
model_unbalanced.fit(X_train_unbal, y_train_unbal)

# testing on unbalanced data
y_pred_unbalanced = model_unbalanced.predict(X_test_unbal)

# Unbalanced model - evaluation
acc_unbalanced = accuracy_score(y_test_unbal, y_pred_unbalanced)
print("Accuracy on unbalanced dataset: {:.2f}%".format(acc_unbalanced * 100)
print(classification_report(y_test_unbal, y_pred_unbalanced))
```

```
Accuracy on unbalanced dataset: 95.00%
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       177
           1       0.84      0.70      0.76        23

    accuracy                           0.95       200
   macro avg       0.90      0.84      0.87       200
weighted avg       0.95      0.95      0.95       200
```

4. Implement K-means Clustering.

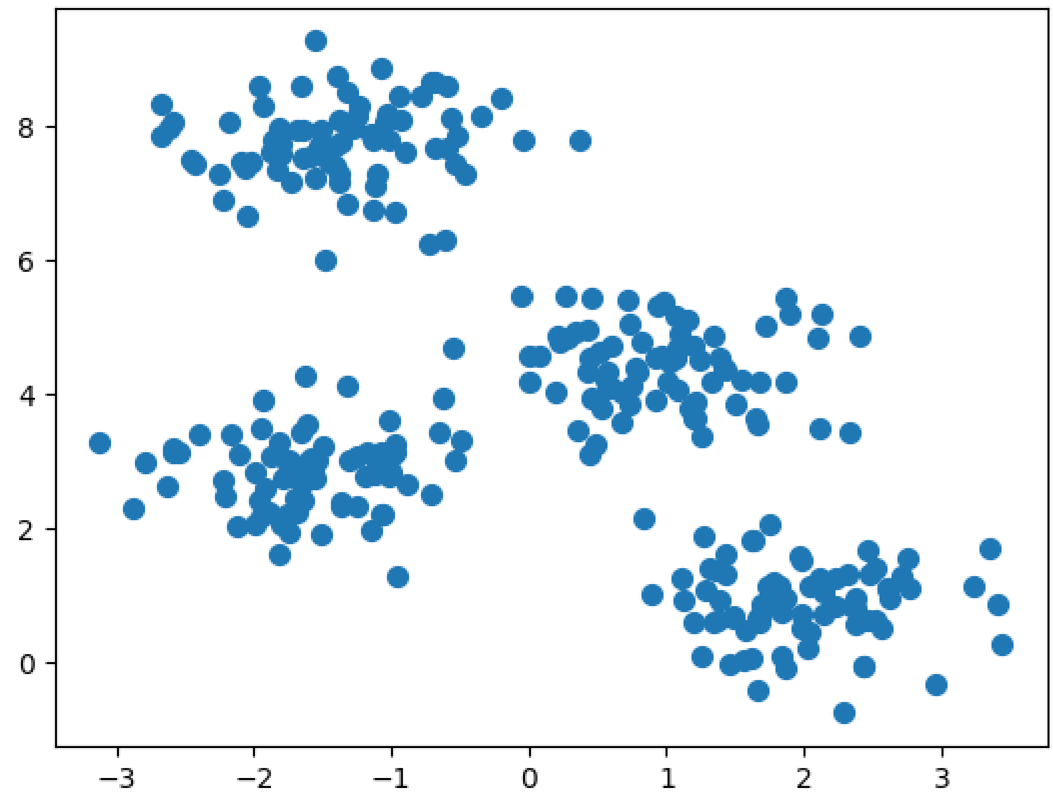
In [5]:

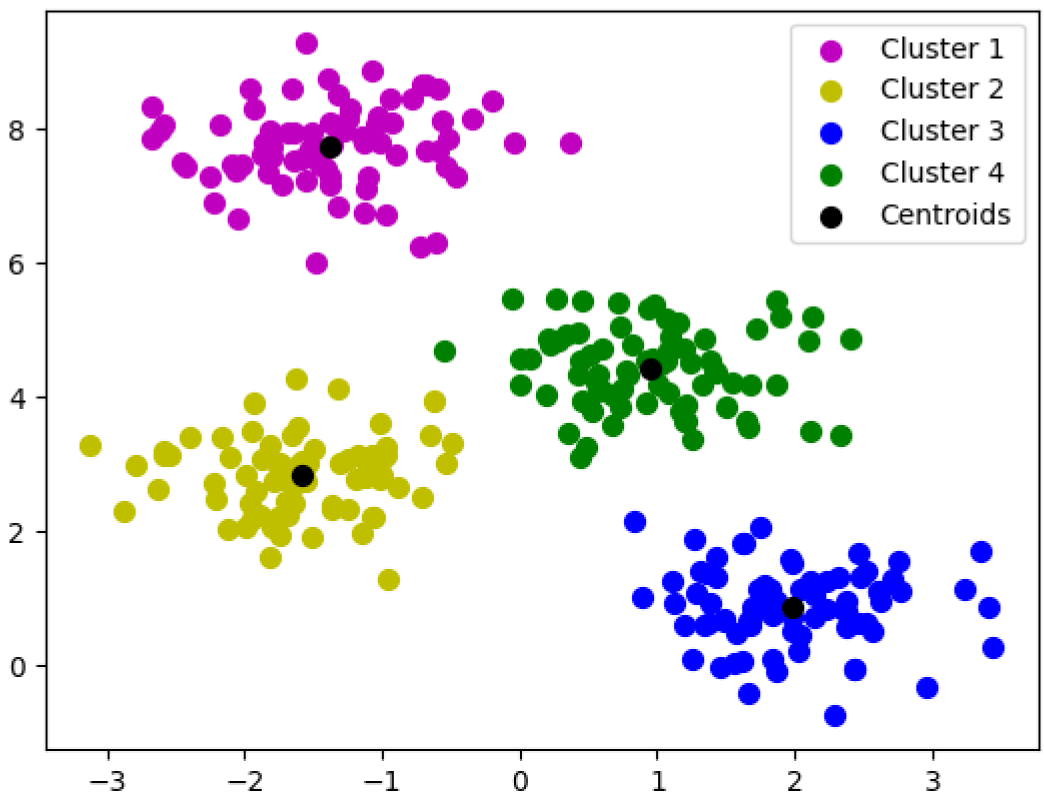
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

In [6]:

```python
# dataset
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0

plt.scatter(X[:, 0], X[:, 1], s=50)
plt.show()

k = 4

kmeans = KMeans(n_clusters=k,n_init=10)

kmeans.fit(X)

centroids = kmeans.cluster_centers_
```

```
# Get the labels of the clusters
labels = kmeans.labels_

# Plot the clusters and the centroids
colors = ['m', 'y', 'b', 'g']
for i in range(k):
    plt.scatter(X[labels == i, 0], X[labels == i, 1], s=50, c=colors[i], lab
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, marker='.', c='black',
plt.legend()
plt.show()
```



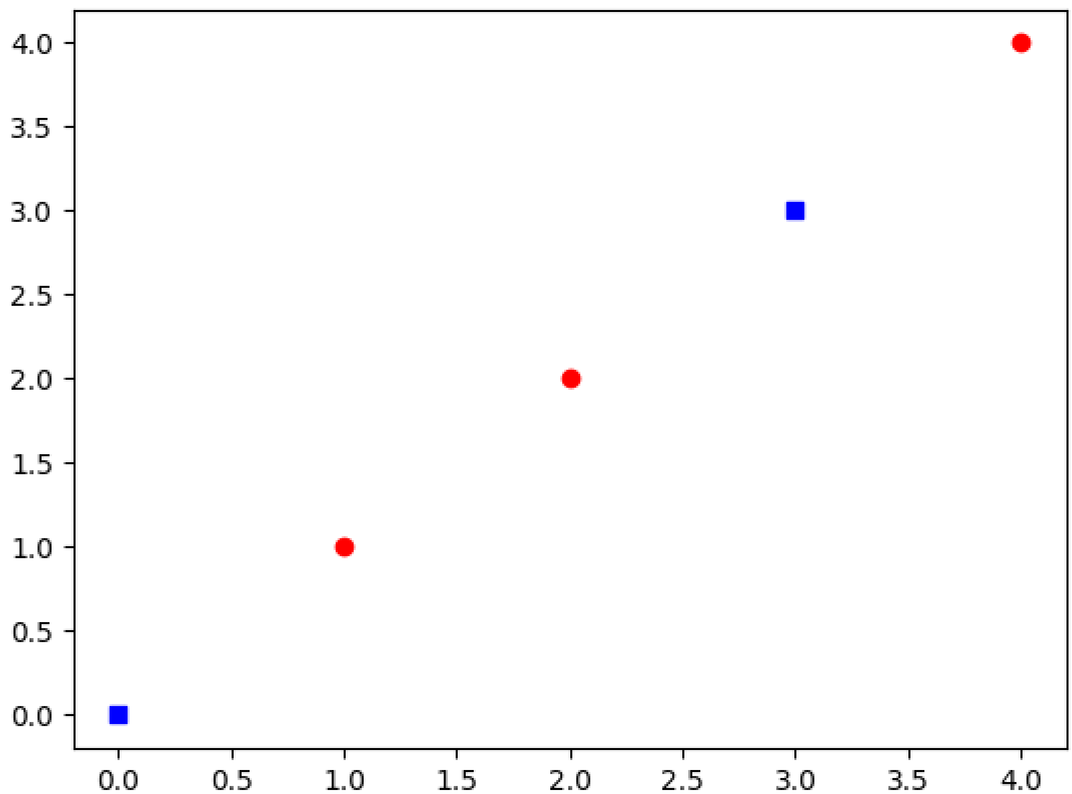


5. Implement K Mode clustering

In [3]:

```
from kmodes.kmodes import KModes
import numpy as np
import matplotlib.pyplot as plt

# random data set
np.random.seed(42)
x = np.array([
    ['Red', 'Small', 'High'],
    ['Blue', 'Medium', 'Low'],
    ['Green', 'Medium', 'Medium'],
    ['Red', 'Large', 'Low'],
    ['Blue', 'Medium', 'Low']
])

# K=2
km = KModes(n_clusters=2, init='Huang', n_init=5, verbose=1)

clusters = km.fit_predict(x)

colors = ['r', 'b']
markers = ['o', 's']
for i in range(len(x)):
    plt.scatter(i, i, color=colors[clusters[i]], marker=markers[clusters[i]])
plt.show()
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 4.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 4.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 4.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 4.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 4.0
Best run was number 1
```



6. Compare the performance of following Machine Learning algorithms

for the **"National Institute of Diabetes and Digestive and Kidney Diseases"** dataset available in the Kaggle database.

1. K-nearest neighbour classifier,
2. Decision tree classifier,

3. Support Vector Machine.

In [8]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

data = pd.read_csv("diabetes.csv")

# Splitting for training and testing
X = data.drop('Outcome', axis=1)
y = data['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# KNN
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# SVM
svm = SVC()
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)

# Print the accuracies of the classifiers
print("Accuracy of K-nearest neighbor classifier:", accuracy_knn)
print("Accuracy of decision tree classifier:", accuracy_dt)
print("Accuracy of support vector machine:", accuracy_svm)
```

```
Accuracy of K-nearest neighbor classifier: 0.6623376623376623
Accuracy of decision tree classifier: 0.7727272727272727
Accuracy of support vector machine: 0.7662337662337663
```

7. Implement Principle Component Analysis for Dimensionality Reduction

In [9]:
```
import numpy as np

class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self, X):
```

```
        self.mean = np.mean(X, axis=0)
        X = X - self.mean

        # Covariance
        cov = np.cov(X.T)
        # Eigen Vectors and Eigen Values of COV matrix
        eigenvalues, eigenvectors = np.linalg.eig(cov)
        # sorting eigen vectors
        indices = np.argsort(eigenvalues)[::-1]
        eigenvectors = eigenvectors[:, indices]
        self.components = eigenvectors[:, :self.n_components]

    def transform(self, X):
        X = X - self.mean
        return np.dot(X, self.components)
```

In [20]:
```
import numpy as np
from sklearn.decomposition import PCA

from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
```

In [21]:
```
# Using PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Comparing
print("Original data shape:", X.shape)
print("Reduced data shape:", X_reduced.shape)
print("\n")
print("Original")
print(X)
print("\n")
print("Reduced")
print(X_reduced)
```

```
Original data shape: (150, 4)
Reduced data shape: (150, 2)


Original
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
```

```
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
```

```
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]


Reduced
[[-2.68412563  0.31939725]
 [-2.71414169 -0.17700123]
 [-2.88899057 -0.14494943]
 [-2.74534286 -0.31829898]
 [-2.72871654  0.32675451]
 [-2.28085963  0.74133045]
 [-2.82053775 -0.08946138]
 [-2.62614497  0.16338496]
 [-2.88638273 -0.57831175]
 [-2.6727558  -0.11377425]
```

```
[-2.50694709  0.6450689 ]
[-2.61275523  0.01472994]
[-2.78610927 -0.235112  ]
[-3.22380374 -0.51139459]
[-2.64475039  1.17876464]
[-2.38603903  1.33806233]
[-2.62352788  0.81067951]
[-2.64829671  0.31184914]
[-2.19982032  0.87283904]
[-2.5879864   0.51356031]
[-2.31025622  0.39134594]
[-2.54370523  0.43299606]
[-3.21593942  0.13346807]
[-2.30273318  0.09870885]
[-2.35575405 -0.03728186]
[-2.50666891 -0.14601688]
[-2.46882007  0.13095149]
[-2.56231991  0.36771886]
[-2.63953472  0.31203998]
[-2.63198939 -0.19696122]
[-2.58739848 -0.20431849]
[-2.4099325   0.41092426]
[-2.64886233  0.81336382]
[-2.59873675  1.09314576]
[-2.63692688 -0.12132235]
[-2.86624165  0.06936447]
[-2.62523805  0.59937002]
[-2.80068412  0.26864374]
[-2.98050204 -0.48795834]
[-2.59000631  0.22904384]
[-2.77010243  0.26352753]
[-2.84936871 -0.94096057]
[-2.99740655 -0.34192606]
[-2.40561449  0.18887143]
[-2.20948924  0.43666314]
[-2.71445143 -0.2502082 ]
[-2.53814826  0.50377114]
[-2.83946217 -0.22794557]
[-2.54308575  0.57941002]
[-2.70335978  0.10770608]
[ 1.28482569  0.68516047]
[ 0.93248853  0.31833364]
[ 1.46430232  0.50426282]
[ 0.18331772 -0.82795901]
[ 1.08810326  0.07459068]
[ 0.64166908 -0.41824687]
[ 1.09506066  0.28346827]
[-0.74912267 -1.00489096]
[ 1.04413183  0.2283619 ]
[-0.0087454  -0.72308191]
[-0.50784088 -1.26597119]
[ 0.51169856 -0.10398124]
[ 0.26497651 -0.55003646]
[ 0.98493451 -0.12481785]
[-0.17392537 -0.25485421]
[ 0.92786078  0.46717949]
```

```
[ 0.66028376 -0.35296967]
[ 0.23610499 -0.33361077]
[ 0.94473373 -0.54314555]
[ 0.04522698 -0.58383438]
[ 1.11628318 -0.08461685]
[ 0.35788842 -0.06892503]
[ 1.29818388 -0.32778731]
[ 0.92172892 -0.18273779]
[ 0.71485333  0.14905594]
[ 0.90017437  0.32850447]
[ 1.33202444  0.24444088]
[ 1.55780216  0.26749545]
[ 0.81329065 -0.1633503 ]
[-0.30558378 -0.36826219]
[-0.06812649 -0.70517213]
[-0.18962247 -0.68028676]
[ 0.13642871 -0.31403244]
[ 1.38002644 -0.42095429]
[ 0.58800644 -0.48428742]
[ 0.80685831  0.19418231]
[ 1.22069088  0.40761959]
[ 0.81509524 -0.37203706]
[ 0.24595768 -0.2685244 ]
[ 0.16641322 -0.68192672]
[ 0.46480029 -0.67071154]
[ 0.8908152  -0.03446444]
[ 0.23054802 -0.40438585]
[-0.70453176 -1.01224823]
[ 0.35698149 -0.50491009]
[ 0.33193448 -0.21265468]
[ 0.37621565 -0.29321893]
[ 0.64257601  0.01773819]
[-0.90646986 -0.75609337]
[ 0.29900084 -0.34889781]
[ 2.53119273 -0.00984911]
[ 1.41523588 -0.57491635]
[ 2.61667602  0.34390315]
[ 1.97153105 -0.1797279 ]
[ 2.35000592 -0.04026095]
[ 3.39703874  0.55083667]
[ 0.52123224 -1.19275873]
[ 2.93258707  0.3555    ]
[ 2.32122882 -0.2438315 ]
[ 2.91675097  0.78279195]
[ 1.66177415  0.24222841]
[ 1.80340195 -0.21563762]
[ 2.1655918   0.21627559]
[ 1.34616358 -0.77681835]
[ 1.58592822 -0.53964071]
[ 1.90445637  0.11925069]
[ 1.94968906  0.04194326]
[ 3.48705536  1.17573933]
[ 3.79564542  0.25732297]
[ 1.30079171 -0.76114964]
[ 2.42781791  0.37819601]
[ 1.19900111 -0.60609153]
```

```
[ 3.49992004  0.4606741 ]
[ 1.38876613 -0.20439933]
[ 2.2754305   0.33499061]
[ 2.61409047  0.56090136]
[ 1.25850816 -0.17970479]
[ 1.29113206 -0.11666865]
[ 2.12360872 -0.20972948]
[ 2.38800302  0.4646398 ]
[ 2.84167278  0.37526917]
[ 3.23067366  1.37416509]
[ 2.15943764 -0.21727758]
[ 1.44416124 -0.14341341]
[ 1.78129481 -0.49990168]
[ 3.07649993  0.68808568]
[ 2.14424331  0.1400642 ]
[ 1.90509815  0.04930053]
[ 1.16932634 -0.16499026]
[ 2.10761114  0.37228787]
[ 2.31415471  0.18365128]
[ 1.9222678   0.40920347]
[ 1.41523588 -0.57491635]
[ 2.56301338  0.2778626 ]
[ 2.41874618  0.3047982 ]
[ 1.94410979  0.1875323 ]
[ 1.52716661 -0.37531698]
[ 1.76434572  0.07885885]
[ 1.90094161  0.11662796]
[ 1.39018886 -0.28266094]]
```