```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn import preprocessing
import plotly.express as px
from sklearn.preprocessing import StandardScaler
%matplotlib inline
plt.style.use('dark_background')
```

```
data = pd.read_csv('/content/drive/MyDrive/Dataset/cancer.csv')
data.drop(['id', 'diagnosis', 'Unnamed: 32'], axis=1, inplace=True) # Droping co
```

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data) # Transforming the data to reduce the v
```

```
def euclideanDistance(x, y):
    squared_d = 0
    for i in range(len(x)):
        squared_d += (x[i] - y[i])**2
    d = np.sqrt(squared_d)
    return d
```

```
class k_medoids:
    def __init__(self, k = 2, max_iter = 300, has_converged = False):
        '''
        Class constructor
        Parameters
        ----------
        - k: number of clusters.
        - max_iter: number of times centroids will move
        - has_converged: to check if the algorithm stop or not
        '''
        self.k = k
        self.max_iter = max_iter
        self.has_converged = has_converged
        self.medoids_cost = []

    def initMedoids(self, X):
        self.medoids = []

        #Starting medoids will be random members from dataset X
        indexes = np.random.randint(0, len(X)-1,self.k)
        self.medoids = X[indexes]
```

```
        for i in range(0,self.k):
            self.medoids_cost.append(0)

    def isConverged(self, new_medoids):
        '''
        new_medoids: the recently calculated medoids to be compared with the cur
        '''
        return set([tuple(x) for x in self.medoids]) == set([tuple(x) for x in n

    def updateMedoids(self, X, labels):
        '''
        labels: a list contains labels of data points
        '''
        self.has_converged = True

        #Store data points to the current cluster they belong to
        clusters = []
        for i in range(0,self.k):
            cluster = []
            for j in range(len(X)):
                if (labels[j] == i):
                    cluster.append(X[j])
            clusters.append(cluster)

        #Calculate the new medoids
        new_medoids = []
        for i in range(0, self.k):
            new_medoid = self.medoids[i]
            old_medoids_cost = self.medoids_cost[i]
            for j in range(len(clusters[i])):

                #Cost of the current data points to be compared with the current
                cur_medoids_cost = 0
                for dpoint_index in range(len(clusters[i])):
                    cur_medoids_cost += euclideanDistance(clusters[i][j], cluste

                #If current cost is less than current optimal cost,
                #make the current data point new medoid of the cluster
                if cur_medoids_cost < old_medoids_cost:
                    new_medoid = clusters[i][j]
                    old_medoids_cost = cur_medoids_cost

            #Now we have the optimal medoid of the current cluster
            new_medoids.append(new_medoid)

        #If not converged yet, accept the new medoids
        if not self.isConverged(new_medoids):
            self.medoids = new_medoids
            self.has_converged = False

    def fit(self, X):
        '''
        X: input data.
        '''
```

```
        '''
        self.initMedoids(X)

        for i in range(self.max_iter):
            #Labels for this iteration
            cur_labels = []
            for medoid in range(0,self.k):
                #Dissimilarity cost of the current cluster
                self.medoids_cost[medoid] = 0
                for k in range(len(X)):
                    #Distances from a data point to each of the medoids
                    d_list = []
                    for j in range(0,self.k):
                        d_list.append(euclideanDistance(self.medoids[j], X[k]))
                    #Data points' label is the medoid which has minimal distance
                    cur_labels.append(d_list.index(min(d_list)))

                    self.medoids_cost[medoid] += min(d_list)

            self.updateMedoids(X, cur_labels)

            if self.has_converged:
                break

        return np.array(self.medoids)


    def predict(self,data):
        '''
        Returns:
        ----------
        pred: list cluster indexes of input data
        '''

        pred = []
        for i in range(len(data)):
            #Distances from a data point to each of the medoids
            d_list = []
            for j in range(len(self.medoids)):
                d_list.append(euclideanDistance(self.medoids[j],data[i]))

            pred.append(d_list.index(min(d_list)))

        return np.array(pred)


model=k_medoids(k=2)
print('Centers found by my model:')
print(model.fit(data_scaled))
```

```
Centers found by my model:
[[-0.35992884 -0.30010986 -0.36161014 -0.42260266  0.21205301 -0.16830779
  -0.62661002 -0.66468889 -0.34179632 -0.40084312 -0.50410578 -0.22108402
  -0.53862262 -0.43974991 -0.58268326 -0.4956362  -0.37081874 -0.5867346
  -0.37077855 -0.35816766 -0.42020976 -0.13959348 -0.45814231 -0.45439078
```

```
  -0.15204891 -0.2555058  -0.47537933 -0.53820271 -0.19697403 -0.26410166]
 [ 1.50888457 -0.10929036  1.48824175  1.45649594  0.89239463  0.76675821
   1.71752864  1.81798191  0.04155274 -0.23356584  0.58010091  0.34680562
   0.3786691   0.60291111 -0.58435     0.15938921  0.35366221 -0.00586178
  -0.40831289 -0.22086242  1.59263243  0.76744644  1.38917538  1.51077984
   0.83425864  0.75213951  1.54197931  1.39161624  0.59088513  0.34048491]]
```

```
y_pred = model.predict(data_scaled) # predicting the clusters using scaled data
```

```
print(y_pred) # prediction of clusters in 0 and 1
```

```
[1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0
 0 0 0 1 1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0
 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1
 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0
 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 0 1 0]
```

```
frame = pd.DataFrame(data)
frame['cluster'] = y_pred
frame['cluster'].value_counts() # Counting of cluster points in each of cluster
```

```
0    405
1    164
Name: cluster, dtype: int64
```

```
dataset = data.copy()
dataset['cluster'] = y_pred
```

```
fig = px.scatter_3d(dataset, x="radius_mean", y="texture_mean", z="perimeter_mea
fig.show() # Ploting the 3D scatter plot
```

```
fig = px.scatter(dataset['radius_mean'], dataset['texture_mean'], color=dataset[
fig.show() # Ploting scatter plot using two feature vectors radius_mean, texture
```