MACHINE LEARNING LAB

EXERCISE :: 8

NAME:: Saptarshi Datta

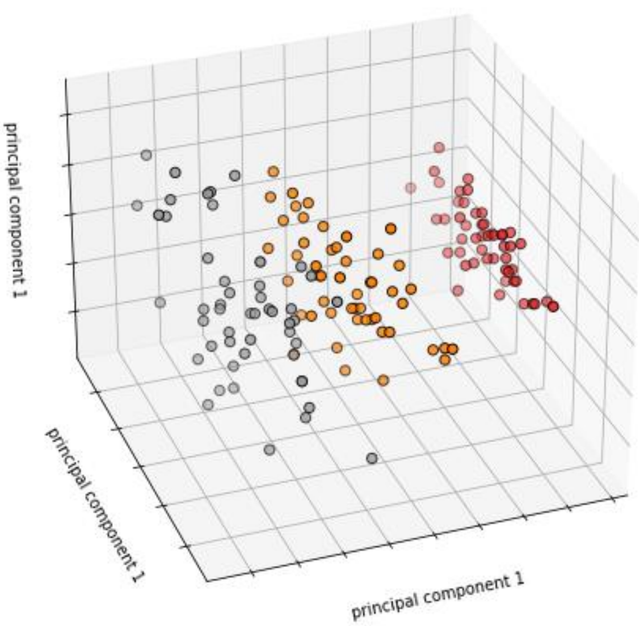
REG NO.:: 19BAI1041

Clustering ::

Code(K-Medoid) ::

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA



# Dataset
iris = datasets.load_iris()
data = pd.DataFrame(iris.data,columns = iris.feature_names)

target = iris.target_names
labels = iris.target

#Scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data = pd.DataFrame(scaler.fit_transform(data), columns=data.columns)



#PCA Transformation
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
principalComponents = pca.fit_transform(data)
```

```
PCAdf = pd.DataFrame(data = principalComponents , columns = ['principal component 1',
'principal component 2','principal component 3'])


datapoints = PCAdf.values
m, f = datapoints.shape
k = 3


#Visualization
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = datapoints
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=labels,
        cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("principal component 1")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("principal component 1")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("principal component 1")
ax.w_zaxis.set_ticklabels([])
plt.show()
```



```
def init_medoids(X, k):
    from numpy.random import choice
    from numpy.random import seed
    seed(1)
    samples = choice(len(X), size=k, replace=False)
    return X[samples, :]
medoids_initial = init_medoids(datapoints, 3)
def compute_d_p(X, medoids, p):
    m = len(X)
    medoids_shape = medoids.shape
    # If a 1-D array is provided,
    # it will be reshaped to a single row 2-D array
    if len(medoids_shape) == 1:
        medoids = medoids.reshape((1,len(medoids)))
    k = len(medoids)
    S = np.empty((m, k))
    for i in range(m):
        d_i = np.linalg.norm(X[i, :] - medoids, ord=p, axis=1)
        S[i, :] = d_i**p
    return S
S = compute_d_p(datapoints, medoids_initial, 2)

def assign_labels(S):
    return np.argmin(S, axis=1)

labels = assign_labels(S)
def update_medoids(X, medoids, p):
    S = compute_d_p(datapoints, medoids, p)
    labels = assign_labels(S)
    out_medoids = medoids
    for i in set(labels):
```

```
        avg_dissimilarity = np.sum(compute_d_p(datapoints, medoids[i], p))
        cluster_points = datapoints[labels == i]
        for datap in cluster_points:
            new_medoid = datap
            new_dissimilarity= np.sum(compute_d_p(datapoints, datap, p))
            if new_dissimilarity < avg_dissimilarity :
                avg_dissimilarity = new_dissimilarity
                out_medoids[i] = datap
    return out_medoids

def has_converged(old_medoids, medoids):
    return set([tuple(x) for x in old_medoids]) == set([tuple(x) for x in medoids])



#Full algorithm
def kmedoids(X, k, p, starting_medoids=None, max_steps=np.inf):
    if starting_medoids is None:
        medoids = init_medoids(X, k)
    else:
        medoids = starting_medoids
    converged = False
    labels = np.zeros(len(X))
    i = 1
    while (not converged) and (i <= max_steps):
        old_medoids = medoids.copy()
        S = compute_d_p(X, medoids, p)
        labels = assign_labels(S)
        medoids = update_medoids(X, medoids, p)
        converged = has_converged(old_medoids, medoids)
        i += 1
    return (medoids,labels)
results = kmedoids(datapoints, 3, 2)
```

```
final_medoids = results[0]
data['clusters'] = results[1]



#Count
def mark_matches(a, b, exact=False):
    """
    Given two Numpy arrays of {0, 1} labels, returns a new boolean
    array indicating at which locations the input arrays have the
    same label (i.e., the corresponding entry is True).
    This function can consider "inexact" matches. That is, if `exact`
    is False, then the function will assume the {0, 1} labels may be
    regarded as the same up to a swapping of the labels. This feature
    allows
      a == [0, 0, 1, 1, 0, 1, 1]
      b == [1, 1, 0, 0, 1, 0, 0]
    to be regarded as equal. (That is, use `exact=False` when you
    only care about "relative" labeling.)
    """
    assert a.shape == b.shape
    a_int = a.astype(dtype=int)
    b_int = b.astype(dtype=int)
    all_axes = tuple(range(len(a.shape)))
    assert ((a_int == 0) | (a_int == 1) | (a_int == 2)).all()
    assert ((b_int == 0) | (b_int == 1) | (b_int == 2)).all()
    exact_matches = (a_int == b_int)
    if exact:
        return exact_matches
    assert exact == False
    num_exact_matches = np.sum(exact_matches)
    if (2*num_exact_matches) >= np.prod (a.shape):
        return exact_matches
```

```
    return exact_matches == False # Invert

def count_matches(a, b, exact=False):
    """
    Given two sets of {0, 1} labels, returns the number of mismatches.
    This function can consider "inexact" matches. That is, if `exact`
    is False, then the function will assume the {0, 1} labels may be
    regarded as similar up to a swapping of the labels. This feature
    allows
    a == [0, 0, 1, 1, 0, 1, 1]
    b == [1, 1, 0, 0, 1, 0, 0]
   to be regarded as equal. (That is, use `exact=False` when you
    only care about "relative" labeling.)
    """
    matches = mark_matches(a, b, exact=exact)
    return np.sum(matches)

n_matches = count_matches(labels, data['clusters'])
print(n_matches,
      "matches out of",
      len(data), "data points",
      "(~ {:.1f}%)".format(100.0 * n_matches / len(labels)))
```

```
142 matches out of 150 data points (~ 94.7%)
```

Code(K-Mean Classifier) ::

```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

```
df = pd.read_csv("Mall_Customers.csv")
df
```

	CustomerID	Genre	Age	Annual_Income_(k$)	Spending_Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   CustomerID          200 non-null    int64
 1   Genre               200 non-null    object
 2   Age                 200 non-null    int64
 3   Annual_Income_(k$)  200 non-null    int64
 4   Spending_Score      200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
df.describe()
```

	CustomerID	Age	Annual_Income_(k$)	Spending_Score
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

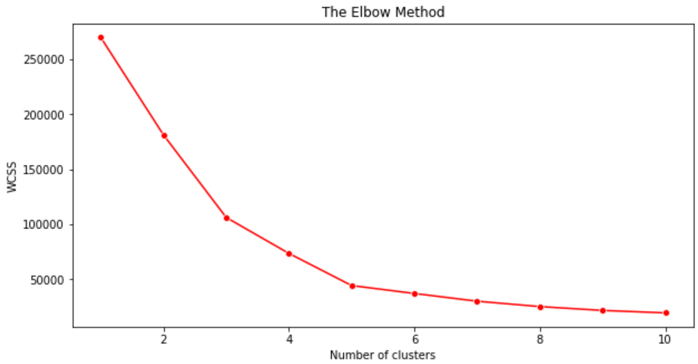
```
X = df.iloc[:, [3, 4]].values
```

```
X
```

```
array([[ 15,  39],[ 15,  81],[ 16,   6],[ 16,  77],[ 17,  40],[ 17,  76],[ 18,   6]
,[ 18,  94],[ 19,   3],[ 19,  72],[ 19,  14],[ 19,  99],[ 20,  15],[ 20,  77],[ 20,
 13],[ 20,  79],[ 21,  35],[ 21,  66],[ 23,  29],[ 23,  98],[ 24,  35],[ 24,  73],
 [ 25,   5],[ 25,  73],[ 28,  14],[ 28,  82],[ 28,  32],[ 28,  61],[ 29,  31],[ 29,
 87],[ 30,   4],[ 30,  73],[ 33,   4],[ 33,  92],[ 33,  14],[ 33,  81],[ 34,  17],[
34,  73],[ 37,  26],[ 37,  75],[ 38,  35],[ 38,  92],[ 39,  36],[ 39,  61],[ 39,  2
8],[ 39,  65],[ 40,  55],[ 40,  47],[ 40,  42],[ 40,  42],[ 42,  52],[ 42,  60],[ 4
3,  54],[ 43,  60],[ 43,  45],[ 43,  41],[ 44,  50],[ 44,  46],[ 46,  51],[ 46,  46
],[ 46,  56],[ 46,  55],[ 47,  52],[ 47,  59],[ 48,  51],[ 48,  59],[ 48,  50],[ 48
,  48],[ 48,  59],[ 48,  47],[ 49,  55],[ 49,  42],[ 50,  49],[ 50,  56],[ 54,  47]
,[ 54,  54],[ 54,  53],[ 54,  48],[ 54,  52],[ 54,  42],[ 54,  51],[ 54,  55],[ 54,
 41],[ 54,  44],[ 54,  57],[ 54,  46],[ 57,  58],[ 57,  55],[ 58,  60],[ 58,  46],[
59,  55],[ 59,  41],[ 60,  49],[ 60,  40],[ 60,  42],[ 60,  52],[ 60,  47],[ 60, 0]
,[ 61,  42],[ 61,  49],[ 62,  41],[ 62,  48],[ 62,  59],[ 62,  55],[ 62,  56],[ 62,
42],[ 63,  50],[ 63,  46],[ 63,  43],[ 63,  48],[ 63,  52][ 63,  54],[ 64,  42],[ 6
4,  46],[ 65,  48],[ 65,  50],[ 65,  43],[ 65,  59],[ 67,  43],[ 67,  57],[ 67,  56
],[ 67,  40],[ 69,  58],[ 69,  91],[ 70,  29],[ 70,  77],[ 71,  35],[ 71,  95],[ 71
,  11],[ 71,  75],[ 71,   9],[ 71,  75],[ 72,  34],[ 72,  71],[ 73,   5],[ 73,  88]
,[ 73,   7],[ 73,  73],[ 74,  10],[ 74,  72],[ 75,   5],[ 75,  93],[ 76,  40],[ 76,
 87],[ 77,  12],[ 77,  97],[ 77,  36],[ 77,  74],[ 78,  22],[ 78,  90],[ 78,  17],[
  78,  88],[ 78,  20],[ 78,  76],[ 78,  16],[ 78,  89],[ 78,   1],[ 78,  78],[ 78,
1],[ 78,  73],[ 79,  35],[ 79,  83],[ 81,   5],[ 81,  93],[ 85,  26],[ 85,  75],[ 8
6,  20],[ 86,  95],[ 87,  27],[ 87,  63],[ 87,  13],[ 87,  75],[ 87,  10],[ 87,  92
],[ 88,  13],[ 88,  86],[ 88,  15],[ 88,  69],[ 93,  14],[ 93,  90],[ 97,  32],[ 97
,  86],[ 98,  15],[ 98,  88],[ 99,  39],[ 99,  97],[101,  24],[101,  68],[103,  17]
,[103,  85],[103,  23],[103,  69],[113,   8],[113,  91],[120,  16],[120,  79],[126,
                        28],[126,  74],[137,  18],[137,  83]])
```

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
   kmeans.fit(X)
   # inertia method returns wcss   for that model
   wcss.append(kmeans.inertia_)

plt.figure(figsize=(10,5))from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
   kmeans.fit(X)
   # inertia method returns wcss for that model
```

```
    wcss.append(kmeans.inertia_)
sns.lineplot(range(1, 11), wcss,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



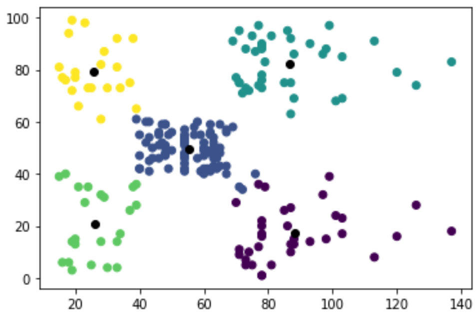

```
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
print(y_kmeans)
```

```
[3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
 4 3 4 3 4 3 1 3 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 0 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2
 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```

```
print(kmeans.cluster_centers_)
```

```
[[88.2        17.11428571]
 [55.2962963  49.51851852]
 [86.53846154 82.12820513]
 [26.30434783 20.91304348]
 [25.72727273 79.36363636]]
```

```
import matplotlib.pyplot as plt

plt.scatter(X[:,0],X[:,1], c=y_kmeans)
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0],centers[:,1], c='black')
```



Code(K-Mode Classifier) ::

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

bank = pd.read_csv("bankmarketing.csv")
bank.head()

bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 809 entries, 0 to 808
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             809 non-null    int64
 1   job             809 non-null    object
 2   marital         809 non-null    object
 3   education       809 non-null    object
 4   default         809 non-null    object
 5   housing         809 non-null    object
 6   loan            809 non-null    object
 7   contact         809 non-null    object
 8   month           809 non-null    object
 9   day_of_week     809 non-null    object
 10  duration        809 non-null    int64
 11  campaign        809 non-null    int64
 12  pdays           809 non-null    int64
 13  previous        809 non-null    int64
 14  poutcome        809 non-null    object
 15  emp.var.rate    809 non-null    float64
 16  cons.price.idx  809 non-null    float64
 17  cons.conf.idx   809 non-null    float64
 18  euribor3m       809 non-null    float64
 19  nr.employed     809 non-null    int64
 20  y               809 non-null    object
dtypes: float64(4), int64(6), object(11)
memory usage: 132.9+ KB
```

```
bank.describe()
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	809.000000	809.000000	809.000000	809.0	809.0	8.090000e+02	8.090000e+02	8.090000e+02	809.000000	809.0
mean	42.619283	267.164400	1.690977	999.0	0.0	1.100000e+00	9.399400e+01	-3.640000e+01	4.856934	5191.0
std	8.693968	231.240893	0.877908	0.0	0.0	1.488619e-14	9.953752e-14	5.047974e-13	0.000248	0.0
min	22.000000	5.000000	1.000000	999.0	0.0	1.100000e+00	9.399400e+01	-3.640000e+01	4.856000	5191.0
25%	36.000000	138.000000	1.000000	999.0	0.0	1.100000e+00	9.399400e+01	-3.640000e+01	4.857000	5191.0
50%	42.000000	209.000000	1.000000	999.0	0.0	1.100000e+00	9.399400e+01	-3.640000e+01	4.857000	5191.0
75%	49.000000	325.000000	2.000000	999.0	0.0	1.100000e+00	9.399400e+01	-3.640000e+01	4.857000	5191.0
max	60.000000	2033.000000	6.000000	999.0	0.0	1.100000e+00	9.399400e+01	-3.640000e+01	4.857000	5191.0

```
bank_cust = bank[['age','job', 'marital', 'education', 'default', 'housing', 'loan','contact','month
','day_of_week','poutcome']]
bank_cust.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	nonexistent
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	nonexistent
2	37	services	married	high.school	no	yes	no	telephone	may	mon	nonexistent
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	nonexistent
4	56	services	married	high.school	no	no	yes	telephone	may	mon	nonexistent

```
bank_cust['age_bin'] = pd.cut(bank_cust['age'], [0, 20, 30, 40, 50, 60, 70, 80, 90, 100],
                              labels=['0-20', '20-30', '30-40', '40-50','50-60','60-70','70-80', '80-90','90-1
00'])
bank_cust  = bank_cust.drop('age',axis = 1)
```

```
bank_cust.head()
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	age_bin
0	housemaid	married	basic.4y	no	no	no	telephone	may	mon	nonexistent	50-60
1	services	married	high.school	unknown	no	no	telephone	may	mon	nonexistent	50-60
2	services	married	high.school	no	yes	no	telephone	may	mon	nonexistent	30-40
3	admin.	married	basic.6y	no	no	no	telephone	may	mon	nonexistent	30-40
4	services	married	high.school	no	no	yes	telephone	may	mon	nonexistent	50-60

```
bank_cust.shape
```

```
(809, 11)
```

```
bank_cust.isnull().sum()*100/bank_cust.shape[0]
```

```
job            0.0
marital        0.0
education      0.0
default        0.0
housing        0.0
loan           0.0
contact        0.0
month          0.0
day_of_week    0.0
poutcome       0.0
age_bin        0.0
dtype: float64
```

```
bank_cust_copy = bank_cust.copy()
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
bank_cust = bank_cust.apply(le.fit_transform)
bank_cust.head()
```

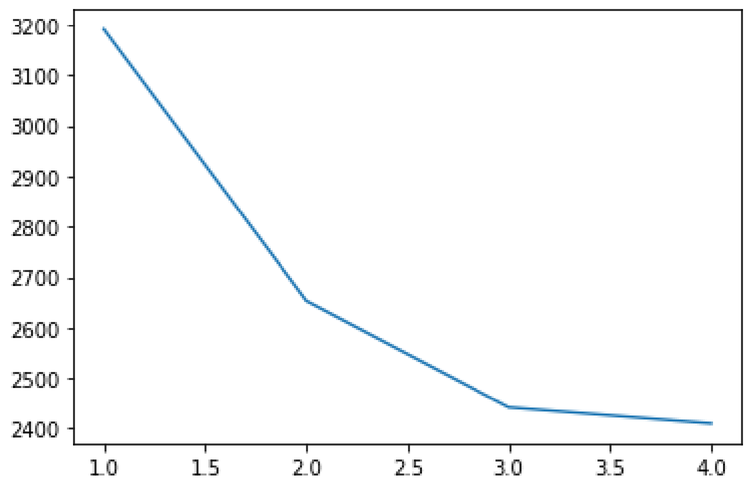
	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	age_bin
0	3	1	0	0	0	0	0	0	0	0	3
1	7	1	3	1	0	0	0	0	0	0	3
2	7	1	3	0	2	0	0	0	0	0	1
3	0	1	1	0	0	0	0	0	0	0	1
4	7	1	3	0	0	2	0	0	0	0	3

```
from kmodes.kmodes import KModes
cost = []
for num_clusters in list(range(1,5)):
    kmode = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    kmode.fit_predict(bank_cust)
    cost.append(kmode.cost_)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 3191.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 144, cost: 2694.0
Run 1, iteration: 2/100, moves: 55, cost: 2653.0
Run 1, iteration: 3/100, moves: 37, cost: 2653.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 172, cost: 2442.0
Run 1, iteration: 2/100, moves: 51, cost: 2442.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 112, cost: 2410.0
```

```
y = np.array([i for i in range(1,5,1)])
plt.plot(y,cost)
```

```
[<matplotlib.lines.Line2D at 0x7fa652de83d0>]
```




```
km_cao = KModes(n_clusters=2, init = "Cao", n_init = 1, verbose=1)
```

fitClusters_cao = km_cao.fit_predict(bank_cust)

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 144, cost: 2694.0
Run 1, iteration: 2/100, moves: 55, cost: 2653.0
Run 1, iteration: 3/100, moves: 37, cost: 2653.0
```