Clustering

Dataset Description

Anuran Calls (MFCCs) Data Set

Acoustic features extracted from syllables of anuran (frogs) calls, including the family, the genus, and the species labels (multilabel).

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:
```
data=pd.read_csv("Frogs_MFCCs.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	MFCCs_1	MFCCs_2	MFCCs_3	MFCCs_4	MFCCs_5	MFCCs_6	MFCCs_7	MFCCs_8	MFCCs_9	MFCCs_10	...	MFCCs_17	MFCCs_18
0	1.0	0.152936	-0.105586	0.200722	0.317201	0.260764	0.100945	-0.150063	-0.171128	0.124676	...	-0.108351	-0.077623
1	1.0	0.171534	-0.098975	0.268425	0.338672	0.268353	0.060835	-0.222475	-0.207693	0.170883	...	-0.090974	-0.056510
2	1.0	0.152317	-0.082973	0.287128	0.276014	0.189867	0.008714	-0.242234	-0.219153	0.232538	...	-0.050691	-0.023590
3	1.0	0.224392	0.118985	0.329432	0.372088	0.361005	0.015501	-0.194347	-0.098181	0.270375	...	-0.136009	-0.177037
4	1.0	0.087817	-0.068345	0.306967	0.330923	0.249144	0.006884	-0.265423	-0.172700	0.266434	...	-0.048885	-0.053074

5 rows × 26 columns

In [4]:
```
X=data.iloc[:,1:-4]
Y=data.iloc[:,-4]
```

In [5]:
```
Y.nunique()
```

Out[5]: 4

In [6]:
```
sc=StandardScaler()
X=sc.fit_transform(X)
```

In [7]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

K-Means Clustering

K-means Clustering (Library)

In [8]:
```
from sklearn.cluster import KMeans
```

In [9]:
```
from sklearn.metrics import silhouette_score

x=[]
sil = []
kmax = 10

for k in range(2, kmax+1):
  kmeans = KMeans(n_clusters = k).fit(X_train)
  labels = kmeans.labels_
  sil.append(silhouette_score(X_train, labels, metric = 'euclidean'))
  x.append(k)
```

In [10]:
```python
for i in zip(x,sil):
  print(i)
```

```
(2, 0.33615042568302983)
(3, 0.3586470357055605)
(4, 0.36090644028848684)
(5, 0.36469802624333525)
(6, 0.2825279468442282)
(7, 0.28995488336593667)
(8, 0.2932980160754407)
(9, 0.29943154667052946)
(10, 0.23809084635969954)
```

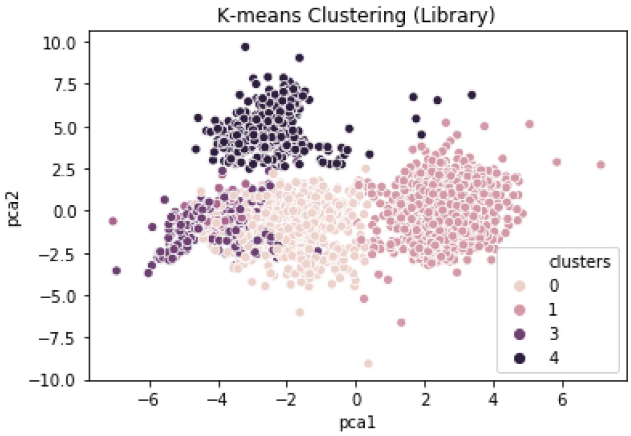
In [11]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
km = KMeans(n_clusters = 5)
```

In [12]:
```python
data=pd.DataFrame(X_train)
data['clusters'] = km.fit_predict(data)
reduced_data = PCA(2).fit_transform(data)
```

In [13]:
```python
reduced_data.shape
```

Out[13]: (5756, 2)

In [14]:
```python
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-means Clustering (Library)')
plt.show()
```



In [15]:
```python
from sklearn.metrics import silhouette_score
silhouette_score(X_train,data['clusters'])
```

Out[15]: 0.3647438484624485

K-Means Clustering (Custom)

(continued on next page)

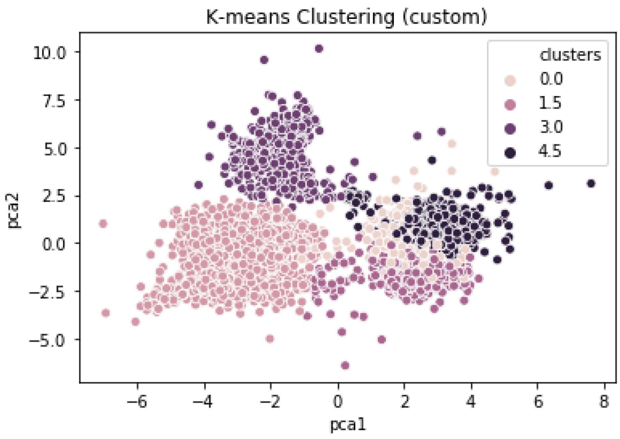
In [16]:
```
class KMeans_custom:
    def __init__(self, n_clusters):
        self.data = pd.DataFrame()
        self.n_clusters = n_clusters
        self.centroids = pd.DataFrame()
        self.clusters = np.ndarray(1)
        self.old_centroids = pd.DataFrame()
        self.verbose = False
        self.predictions = list()

    def train(self, df, verbose):
        self.verbose = verbose
        self.data = df.copy(deep=True)
        self.clusters = np.zeros(len(self.data))

        if 'species' in self.data.columns:
            self.data.drop('species', axis=1, inplace=True)

        unique_rows = self.data.drop_duplicates()
        unique_rows.reset_index(drop=True, inplace=True)
        self.centroids = unique_rows.sample(n=self.n_clusters)
        self.centroids.reset_index(drop=True, inplace=True)

        if self.verbose:
            print("\nRandomly initiated centroids:")
            print(self.centroids)

        self.old_centroids = pd.DataFrame(np.zeros(shape=(self.n_clusters, self.data.shape[1])),
                                          columns=self.data.columns)

        while not self.old_centroids.equals(self.centroids):

            if self.verbose:
                time.sleep(3)

            self.old_centroids = self.centroids.copy(deep=True)

            for row_i in range(0, len(self.data)):
                distances = list()
                point = self.data.iloc[row_i]

                for row_c in range(0, len(self.centroids)):
                    centroid = self.centroids.iloc[row_c]
                    distances.append(np.linalg.norm(point - centroid))

                self.clusters[row_i] = np.argmin(distances)

            for cls in range(0, self.n_clusters):

                cls_idx = np.where(self.clusters == cls)[0]

                if len(cls_idx) == 0:
                    self.centroids.loc[cls] = self.old_centroids.loc[cls]
                else:
                    # Set the new k-mean to the mean value of the data points within this cluster
                    self.centroids.loc[cls] = self.data.iloc[cls_idx].mean()

                if self.verbose:
                    print("\nRow indices belonging to cluster {}: [n={}]".format(cls, len(cls_idx)))
                    print(cls_idx)

            if self.verbose:
                print("\nOld centroids:")
                print(self.old_centroids)
                print("New centroids:")
                print(self.centroids)
```

In [17]:
```
km = KMeans_custom(n_clusters=5)
```

In [18]:
```
data=pd.DataFrame(X_train)
km.train(data,verbose=False)
```

In [19]:
```
data['clusters'] = km.clusters
reduced_data = PCA(2).fit_transform(data)
```

In [20]:
```
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-means Clustering (custom)')
plt.show()
```



In [21]:
```
silhouette_score(X_train,data['clusters'])
```

Out[21]:
```
0.19872144546341883
```

Inference

```
K-Means Clustering (Library) Silhouette Score : 0.36 where n=5
K-Means Clustering (Custom)  Silhouette Score : 0.19 where n=5
```

Lower silhouette score of the custom algorithm indicates more overlapping of the clusters, thereby the library function gets a better score due to it's optimised techniques.

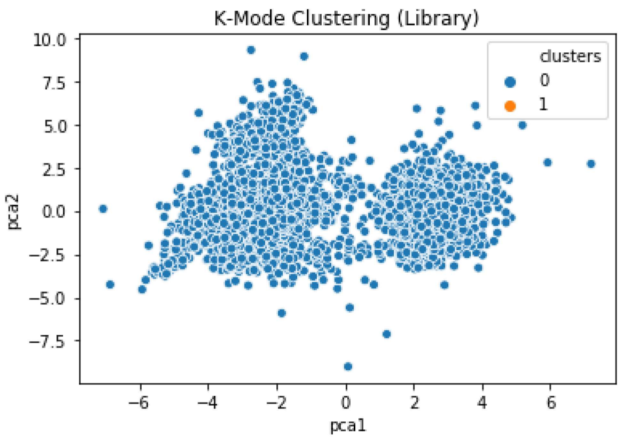
K-Mode Clustering

K-Mode Clustering (Library)

In [22]:
```
from kmodes.kmodes import KModes
kmo = KModes(n_clusters = 2)
```

In [23]:
```
data=pd.DataFrame(X_train)
data['clusters'] = kmo.fit_predict(data)
reduced_data = PCA(2).fit_transform(data)
```

In [24]:
```
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-Mode Clustering (Library)')
plt.show()
```



In [25]:
```python
silhouette_score(X_train,data['clusters'])
```

Out[25]: 0.18575828514490955

K-Mode Clustering (Custom)

In [26]:
```python
from sklearn.base import BaseEstimator, ClusterMixin
from sklearn.utils import check_random_state
from sklearn.utils.validation import check_array
```

In [27]:
```python
from util import get_max_value_key, encode_features, get_unique_rows, decode_centroids, pandas_to_numpy
from util.dissim import matching_dissim, ng_dissim
```

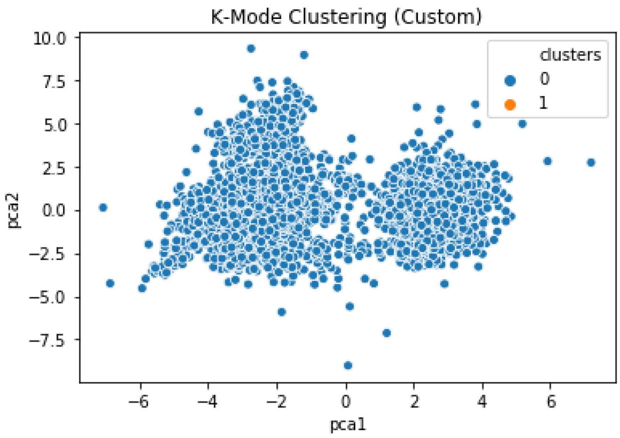
In [28]:
```python
from helper import init_huang,init_cao,move_point_cat,_labels_cost,_k_modes_iter,k_modes,k_modes_single
```

In [29]:
```python
class KModes_Custom(BaseEstimator, ClusterMixin):
    def __init__(self, n_clusters=8, max_iter=100, cat_dissim=matching_dissim,
                 init='Cao', n_init=1, verbose=0, random_state=None, n_jobs=1):

        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.cat_dissim = cat_dissim
        self.init = init
        self.n_init = n_init
        self.verbose = verbose
        self.random_state = random_state
        self.n_jobs = n_jobs
        if ((isinstance(self.init, str) and self.init == 'Cao') or
                hasattr(self.init, '__array__')) and self.n_init > 1:
            if self.verbose:
                print("Initialization method and algorithm are deterministic. "
                      "Setting n_init to 1.")
            self.n_init = 1

    def fit(self, X, y=None, **kwargs):
        X = pandas_to_numpy(X)

        random_state = check_random_state(self.random_state)
        self._enc_cluster_centroids, self._enc_map, self.labels_, self.cost_, \
        self.n_iter_, self.epoch_costs_ = k_modes(
            X,
            self.n_clusters,
            self.max_iter,
            self.cat_dissim,
            self.init,
            self.n_init,
            self.verbose,
            random_state,
            self.n_jobs,
        )
        return self

    def fit_predict(self, X, y=None, **kwargs):
        return self.fit(X, **kwargs).predict(X, **kwargs)

    def predict(self, X, **kwargs):
        assert hasattr(self, '_enc_cluster_centroids'), "Model not yet fitted."
        X = pandas_to_numpy(X)
        X = check_array(X, dtype=None)
        X, _ = encode_features(X, enc_map=self._enc_map)
        return _labels_cost(X, self._enc_cluster_centroids, self.cat_dissim)[0]

    @property
    def cluster_centroids_(self):
        if hasattr(self, '_enc_cluster_centroids'):
            return decode_centroids(self._enc_cluster_centroids, self._enc_map)
        else:
            raise AttributeError("'{}' object has no attribute 'cluster_centroids_' "
                                 "because the model is not yet fitted.")
```

In [30]:
```python
kmo = KModes_Custom(n_clusters = 2)
```

In [31]:
```python
data=pd.DataFrame(X_train)
data['clusters'] = kmo.fit_predict(data)
reduced_data = PCA(2).fit_transform(data)
```

In [32]:
```
results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])
sns.scatterplot(x="pca1", y="pca2", hue=data['clusters'], data=results)
plt.title('K-Mode Clustering (Custom)')
plt.show()
```



In [34]:
```
silhouette_score(X_train,data['clusters'])
```

Out[34]: 0.11330828514490955

Inference

```
K-Modes Clustering (Library) Silhouette Score : 0.18 where n=2
K-Modes Clustering (Custom)  Silhouette Score : 0.11 where n=2
```

- The low performance of k-mode clustering algorithms is due to the fact that k-mode is more suited for categorical variables and this dataset lacks categorical variables.
- Lower silhouette score of the custom algorithm indicates more overlapping of the clusters, thereby the library function gets a better score due to it's optimised techniques.