CISC/CMPE 452/COGS 400 Assignment 3 - Unsupervised Learning (10 points)

Please put your name and student id

```
FirstName LastName, #12345678
```

- Make sure to run all the cells from the beginning before submission. Do not clear out the outputs. You will only get credit for code that has been run.
- Mark will be deducted based on late policy (-1% of the course total marks per day after due date until the end date after which no assignments will be accepted).
- You can only use Numpy to build the models. Other packages such as Pandas, Sklearn and Scipy can be used for evaluation metrics calculating, data processing, and file reading and writing.

Files need to be uploaded for this assignment: A3.ipynb, output.wav, and output.csv

In [ ]:

Part 1 Principle Component Analysis Network (5 points)

The dataset "data/sound.csv" contains two sounds recorded by the two microphones. The goal of this assignment is using PCA network to find the approximation of the first principal component.

- Build a PCA network (refer to Principal Component Analysis slide #22 and #23) to reduce the number of features from 2 to 1 (3 points)
- Train the model and generate the processed data (1 point)
- Save the data into output.wav and output.csv files (1 point)
- Compare the sound_o.wav (audio with noise) and output.wav (audio is denoised)

In [ ]:
```
import numpy as np
import pandas as pd
from scipy.io import wavfile
```

In [ ]:
```
samrate = 8000
```

In [ ]:
```
# read csv into array
txtData = np.genfromtxt('data/sound.csv', delimiter=',')
txtData.shape
```

In [ ]:
```
# save array to WAV file
scaledData = np.int16(txtData * samrate)
wavfile.write('data/sound_o.wav', samrate, scaledData)
```

In [ ]:
```
# read WAV file into array
# The data in sound.csv is processed
# If you use the data generated here, you need to process the data by
samrate, wavData = wavfile.read('data/sound_o.wav')
samrate, wavData.shape
```

In [ ]:
```
# save array to csv file
np.savetxt('data/sound_o.csv', txtData, delimiter=',')
```

In [ ]:
```
# build PCA model and only Numpy can be used
class PCA(object):
    def __init__(self, lr, epoch):

    def train(self, x, n_components):
```

In [ ]:
```
# initialize and train the model
```

In [ ]:
```
# save the data
```

Part 2 K-Means Clustering Algorithm (5 points)

The dataset is Palmer Archipelago (Antarctica) penguin data (https://www.kaggle.com/datasets/parulpandey/palmer-archipelago-antarctica-penguin-data) which has 6 features and 1 label called species (Chinstrap, Adélie, or Gentoo)
The dataset is saved in the "data/penguins_size.csv" file and preprocessed into x_train, x_test, y_train, y_test

- Build a K-Means clustering algorithm (refer to Unsupervised Learning slide #29) to cluster the preprocessed data (2 points)
- Standardize the data and train the model with the training set (1 point)
- Evaluate the model and print the confusion matrixes with both training and test sets (2 points)

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [ ]:
```
# load the dataset
data = pd.read_csv('data/penguins_size.csv')
data.head()
```

In [ ]:
```
# data preprocessing
data = data.dropna()
data = data[data['sex'] != '.']

cleanup_nums = {"species": {"Adelie": 0, "Chinstrap": 1, "Gentoo": 2},
                "island": {"Biscoe": 0, "Dream": 1, "Torgersen": 2},
                "sex": {"MALE": 0.0, "FEMALE": 1.0}}
data = data.replace(cleanup_nums)

data.head()
```

In [ ]:
```
x = np.array(data.drop(['species'], axis=1).copy())
y = np.array(data['species'].copy()).astype(int)
```

In [ ]:
```
# data standardization
x =
```

In [ ]:
```
# split the dataset
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

In [ ]:
```
# calculate the confusion matrix
def evaluator(y, y_pred):
    confusion_matrix =
    print('Confusion matrix:\n', confusion_matrix)
```

In [ ]:
```
# setup a baseline model
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3) # n_clusters - the number of clusters
km.fit(x_train)
y_pred = km.predict(x_train)
evaluator(y_train, y_pred)
y_pred = km.predict(x_test)
evaluator(y_test, y_pred)
```

In [ ]:
```

```

In [ ]:
```
# build K-means model and only Numpy can be used
class KMeans(object):
    def __init__(self):

    def train(self, x, y, x_test, y_test, learning_rate, n_iters):

    def predict(self, x):

```

In [ ]:
```
# initialize and train the model

```

In [ ]:
```
# evaluate the model and print the confusion matrixes for both trainin

```

In [ ]:
```

```

In [ ]:
```

```