D A Y S

Build K-mean cluster in python:

A cluster refers to a collection of data points aggregated together because of certain similarities.

Step 1: Import libraries

```
import numpy as np
import matplotlib.pyplot as plt
```

- Numpy for carrying out efficient computations
- Matplotlib for visualization of data

Step 2: Generate random data and Create K-Means Algorithm

Generate random data normally distributed around 3 centers

```
def generate_data(std=1, dim=2, dist=4):
    mu0 = np.array([0,0])
    mu1 = np.array([dist, dist])
    mu2 = np.array([0, dist])
    Nc = 200 # for num of data
    x0 = np.random.randn(Nc, dim) * std + mu0
    x1 = np.random.randn(Nc, dim) * std + mu1
    x2 = np.random.randn(Nc, dim) * std + mu2
    x = np.concatenate((x0, x1, x2), axis=0)
    return x


def plot_k_means(x, y, k):

    random_colors = np.random.random((k, 3))
    colors = y.dot(random_colors)
    print(y[:5])
    plt.scatter(x[:,0], x[:,1], c=colors)
    plt.title("40 Days")
    plt.show()
```

Build K-mean cluster in python:

Step 3: Finding the centroid and calculate the distance between data points

```
def initialize_centroids(x, num_k):
    N, D = x.shape
    centroids= np.zeros((num_k, D))
    used_idx = []
    for k in range(num_k):
        idx = np.random.choice(N)
        while idx in used_idx:
            idx = np.random.choice(N)
        used_idx.append(idx)
        centroids[k] = x[idx]
    return centroids

def update_centroids(x, y, K):
    N, D = x.shape
    centroids = np.zeros((K, D))    # old center
    for k in range(K):
        centroids[k] = y[:, k].dot(x) / y[:, k].sum()
    return centroids


def square_dist(a, b):
    return (a - b) ** 2

def cost_func(x, y, centroids, K):

    cost = 0
    for k in range(K):
        norm = np.linalg.norm(x - centroids[k], 2)
        cost += (norm * np.expand_dims(y[:, k], axis=1) ).sum()
    return cost


def cluster_responsibilities(centroids, x, beta):
    N, _ = x.shape
    K, D = centroids.shape
    R = np.zeros((N, K))

    for n in range(N):
        R[n] = np.exp(-beta * np.linalg.norm(centroids - x[n], 2, axis=1))
    R /= R.sum(axis=1, keepdims=True)
```

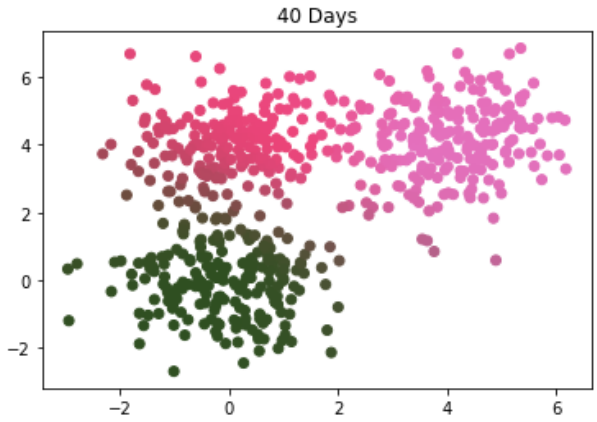
Visualizing The Clusters and set iteration

```
def k_means(x, K, max_iters=5, beta=1.):
    centroids = initialize_centroids(x, K)
    prev_cost = 0
    for _ in range(max_iters):
        y = cluster_responsibilities(centroids, x, beta)
        centroids = update_centroids(x, y, K)
        cost = cost_func(x, y, centroids, K)
        if np.abs(cost - prev_cost) < 1e-5:
            break
        prev_cost = cost

    plot_k_means(x, y, K)


def main():
    x = generate_data()
    k_means(x, K=3 )
```

Build K-mean cluster in python:

The output:

```
[[0.96713624 0.00911193 0.02375183]
 [0.85357089 0.07501784 0.07141127]
 [0.96942289 0.00934856 0.02122855]
 [0.94894694 0.02075447 0.0302986 ]
 [0.94017989 0.02381148 0.03600863]]
```



If i miss some important part of code check here Github repo:

Link