In [1]:

```
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns;
from scipy.spatial import distance
from sklearn.preprocessing import StandardScaler
import time
import math
sns.set()
import numpy as np
from collections import defaultdict
from sklearn.cluster import KMeans
from numpy import dot
from numpy.linalg import norm
from scipy import spatial
```

In [2]:

```
#Euclidean distance
def euclidean_distance(point1, point2):
    distance = 0
    for a,b in zip(point1, point2):
        distance += pow((a-b), 2)
    return math.sqrt(distance)
```

In [3]:

```
#Cosine Similarity
def cosine_similarity(point1, point2):
  A = np.array(point1)
  B = np.array(point2)
  dist = 1 - np.dot(A,B)/(np.linalg.norm(A)*np.linalg.norm(B))
  return dist
```

In [4]:

```
#jaccard
def jaccard(A, B):
    return 1 - (np.sum(np.minimum(A,B), axis = 0)/np.sum(np.maximum(A, B),
```

In [5]:

```
#Calculating centroid
def calculate_centroid(cluster):
    n = len(cluster[0])
    if isinstance(cluster[0][-1], str):
        centroid = [0] * (n - 1)

        for i in range(n - 1):
            for point in cluster:
                centroid[i] += point[i]
            centroid[i] = centroid[i] / len(cluster)
    else:
        centroid = [0] * n

        for i in range(n):
            for point in cluster:
                centroid[i] += point[i]
            centroid[i] = centroid[i] / len(cluster)

    return centroid
```

In [6]:

```
#plotting the clusters
def draw_and_scatter(clusters, centroid_centers):
    colors = ["red", "blue", "green"]
    for i, key in enumerate(clusters):
        x = []
        y = []
        cluster = clusters[key]
        for c in cluster:
            x.append(c[0])
            y.append(c[1])
        plt.scatter(x, y, marker='^', c=colors[i])

    for point in centroid_centers:
        plt.scatter(point[0], point[1], marker='s')

    plt.show()
```

In [7]:

```
def label_cluster(cluster):
  cl = defaultdict(int)
  for point in cluster:
    cl[point[-1]] += 1
  return cl
```

In [8]:

```
# Implementing KMeans
class KMeans:
    def __init__(self, n_clusters=10, max_iters=10, init_centroids=None, d_
                 show_first_centroid=False, centroid_stop=True):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        self.init_centroids = init_centroids
        self.d_func = d_func
        self.sse_list = []
        self.show_first_centroid = show_first_centroid
        self.show_sse = show_sse

    def fit(self, data):
        start = time.time()
        if self.init_centroids is None:
            # Assign random points of data as centroids of size k (n_cluste
            random_choice = np.random.choice(range(len(data)), self.n_clust
            centroids = []

            for choice in random_choice:
                if isinstance(data[choice][-1], str):
                    centroids.append(data[choice][:-1])
                else:
                    centroids.append(data[choice])

            self.init_centroids = centroids

        for loop in range(self.max_iters):
            print("Running: ",loop)
            clusters = defaultdict(list)
            sse = 0
            # Now, assign each point to nearest centroid cluster

            for point in data:
                temp_centroid = -1
                min_dist = 99999999
                for i, centroid in enumerate(self.init_centroids):
                    if isinstance(point[-1], str):
                        d = self.d_func(point[:-1], centroid)
                    else:
                        d = self.d_func(point, centroid)
                    if d < min_dist:
                        temp_centroid = i
                        min_dist = d

                clusters[temp_centroid].append(point)

            prev_centroids = self.init_centroids.copy()
            # Now, recalculating the centroids
            for key in clusters.keys():
                cluster = clusters[key]
                self.init_centroids[key] = calculate_centroid(cluster)

            if loop == 1 and self.show_first_centroid == True:
                print("Centroids after first iteration: ", self.init_centro
            if self.init_centroids == prev_centroids:
                break

            for key in clusters.keys():
                cluster = clusters[key]
                ce = self.init_centroids[key]

                for p in cluster:
                    sse += euclidean_distance(ce, p)

            if self.show_sse == True and loop > 1 and self.sse_list[-1] <=
                self.sse_list.pop()
                break

            self.sse_list.append(sse)

        print("Time taken:", time.time() - start)
        print("Number of iterations:", loop)
        return [self.init_centroids, clusters]
```

In [10]:

```
label = pd.read_csv("label.csv").to_numpy()
data = pd.read_csv("data.csv").to_numpy()
```

In [11]:

```
arr = []

for row in range(len(data)):
  temp = []
  for col in range(len(data[row])):
    temp.append(data[row][col])
  temp.append(label[row][0])
  arr.append(temp)

arr=sorted(arr, key=lambda x: x[len(arr[0])-1], reverse=False)


target_labels = dict(label_cluster(arr))
print(target_labels)
```

```
{0: 980, 1: 1135, 2: 1032, 3: 1010, 4: 982, 5: 892, 6: 958, 7: 1027, 8: 9
74, 9: 1009}
```

In [12]:

```
def run(func):
    if(func is None):
        kmeans = KMeans()
    else:
        kmeans = KMeans(d_func=func)

    [centroid_centers, clusters] = kmeans.fit(arr)

    labels = {0: 0, 1: 0, 2: 0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0}

    for key in clusters:
      d = dict(label_cluster(clusters[key]))
      mx = 0
      s = 0
      label = ""
      for k in d:
        s += d[k]
        if d[k] > mx:
          mx = d[k]
          label = k
      labels[label] = mx

    #draw_and_scatter(clusters, centroid_centers)

    print("SSE =",kmeans.sse_list)
    print("Original Labels: ", target_labels)
    print("Predicted Labels: ", labels)

    total = 0
    mismatch = 0

    for l in target_labels:
      total += target_labels[l]
      mismatch += abs(target_labels[l] - labels[l])

    accuracy = (total - mismatch) / total

    print("Accuracy =",accuracy)
```

In [13]:

```
print("******* EUCLEDIAN *******")
run(None)
print("******* COSINE *******")
run(cosine_similarity)
print("******* JACCARD *******")
run(jaccard)
```

```
******* EUCLEDIAN *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
Running:  7
Running:  8
Running:  9
Time taken: 1412.5303628444672
Number of iterations: 9
SSE = [16577280.97693797, 16106045.158187386, 15929240.889511475, 1586222
7.126402687, 15831673.112918103, 15814068.983201597, 15799132.01754195, 1
5786598.443646502, 15775489.0893925, 15765052.513618972]
Original Labels:  {0: 980, 1: 1135, 2: 1032, 3: 1010, 4: 982, 5: 892, 6:
958, 7: 1027, 8: 974, 9: 1009}
Predicted Labels:  {0: 236, 1: 622, 2: 594, 3: 667, 4: 530, 5: 0, 6: 759,
7: 600, 8: 414, 9: 0}
Accuracy = 0.44224422442244227
******* COSINE *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
Running:  7
Running:  8
Running:  9
Time taken: 117.59274911880493
Number of iterations: 9
SSE = [16503775.047444513, 16018308.172644345, 15852983.423128223, 157895
81.03980296, 15756025.572445994, 15735603.473746805, 15721328.209045235,
15710566.284579532, 15702990.319493629, 15695144.17609017]
Original Labels:  {0: 980, 1: 1135, 2: 1032, 3: 1010, 4: 982, 5: 892, 6:
958, 7: 1027, 8: 974, 9: 1009}
Predicted Labels:  {0: 235, 1: 660, 2: 783, 3: 761, 4: 490, 5: 0, 6: 687,
7: 689, 8: 479, 9: 0}
Accuracy = 0.47844784478447844
******* JACCARD *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
```

```
Running:  7
Running:  8
Running:  9
Time taken: 185.2474410533905
Number of iterations: 9
SSE = [16756162.55863799, 16084564.413899563, 15868394.854431434, 1577588
0.634826211, 15733129.915630057, 15711002.963216748, 15698368.470048891,
15692469.483441373, 15690680.108877506, 15691633.991212199]
Original Labels:  {0: 980, 1: 1135, 2: 1032, 3: 1010, 4: 982, 5: 892, 6:
958, 7: 1027, 8: 974, 9: 1009}
Predicted Labels:  {0: 884, 1: 645, 2: 759, 3: 806, 4: 221, 5: 0, 6: 736,
7: 686, 8: 303, 9: 443}
Accuracy = 0.5483548354835484
```

In [14]:

```python

print("******* EUCLEDIAN *******")
kmeans = KMeans(centroid_stop=True)
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
print("******* COSINE *******")
kmeans = KMeans(centroid_stop=True, d_func=cosine_similarity)
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
print("******* JACCARD *******")
kmeans = KMeans(centroid_stop=True, d_func=jaccard)
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
```

```
PART A
******* EUCLEDIAN *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
Running:  7
Running:  8
Running:  9
Time taken: 1452.806452035904
Number of iterations: 9
[16473259.420872388, 16029519.52425325, 15916007.211980855, 15857673.89
9813233, 15821754.421939583, 15798231.046579242, 15782429.873353254, 15
769709.576061694, 15758217.00605722, 15751600.248485193]
******* COSINE *******
Running:  0
Running:  1
```

In [15]:

```
print(" PART B ")
print("******* EUCLEDIAN *******")
kmeans = KMeans(show_sse=True)
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
print("******* COSINE *******")
kmeans = KMeans(show_sse=True, d_func=cosine_similarity)
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
print("******* JACCARD *******")
kmeans = KMeans(show_sse=True, d_func=jaccard)
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
```

```
 PART B
******* EUCLEDIAN *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
Running:  7
Running:  8
Running:  9
Time taken: 1423.339868068695
Number of iterations: 9
[16696254.16841901, 16168402.924244296, 15963336.497498557, 15866798.4888
74575, 15822984.504805153, 15801280.66327987, 15788253.403940605, 1577842
2.235685347, 15770275.21919023, 15763480.89917195]
******* COSINE *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
Running:  7
Running:  8
Running:  9
Time taken: 116.84105706214905
Number of iterations: 9
[16394773.67458613, 16005728.553528119, 15873291.549990777, 15806012.8362
97585, 15773884.777602537, 15752146.786590435, 15743523.037878798, 157376
95.285131153, 15732695.98151073, 15729117.016123503]
******* JACCARD *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
Running:  7
Running:  8
```

```
Running:  9
Time taken: 183.68019723892212
Number of iterations: 9
[16730715.22049074, 16081244.739890171, 15933262.639259389, 15883591.0970
08165, 15857464.722615218, 15837971.747076752, 15823425.075585645, 158066
27.832437683, 15791565.097292745, 15776402.195384094]
```

In [ ]:

```
print("********************* PART C *********************")
print("******* EUCLEDIAN *******")
kmeans = KMeans(max_iters=100, show_sse=False, centroid_stop=False)
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
print("******* COSINE *******")
kmeans = KMeans(max_iters=100, show_sse=False, centroid_stop=False, d_func=
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
print("******* JACCARD *******")
kmeans = KMeans(max_iters=100, show_sse=False, centroid_stop=False, d_func=
[centroid_centers, clusters] = kmeans.fit(arr)
print(kmeans.sse_list)
```

```
********************* PART C *********************
******* EUCLEDIAN *******
Running:  0
Running:  1
Running:  2
Running:  3
Running:  4
Running:  5
Running:  6
Running:  7
Running:  8
Running:  9
Running:  10
Running:  11
Running:  12
Running:  13
Running:  14
Running:  15
Running:  16
Running:  17
```

In [ ]: