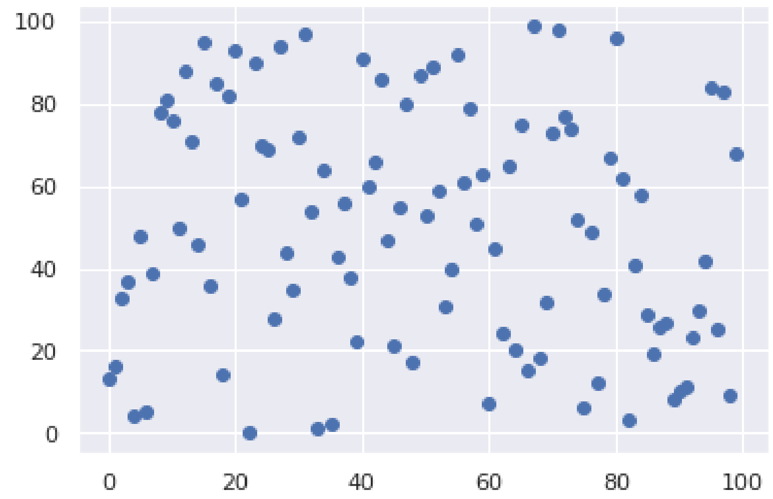
```
# Importing Library
import random
import matplotlib.pyplot as plt
import seaborn as sns
```

```
random.seed(117)
sns.set()
```

```
# Creating a random dataset
x = random.sample(range(0, 100), 100)
y = random.sample(range(0, 100), 100)
dataset = [(i,j) for i,j in zip(x,y)]
```

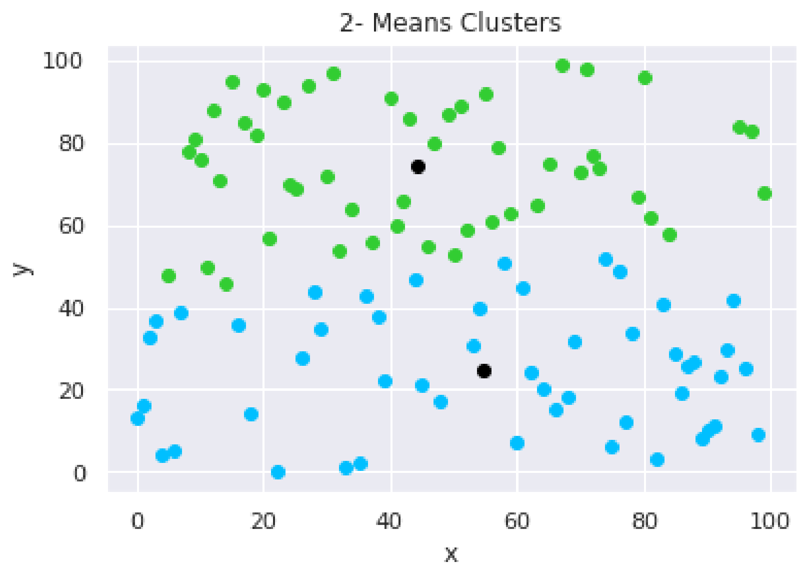
```
plt.scatter(x,y)
plt.show()
```



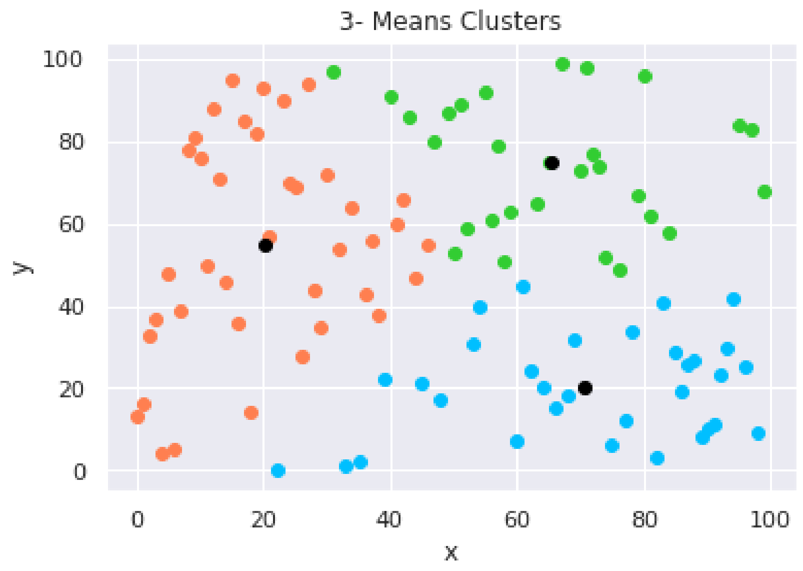
```
# Class for KMeansClustering algorithm
class KMeansClustering():
  def __init__(self, dataset, k, innerIterations, outerIterations):
    self.k = k
    self.innerIterations = innerIterations
    self.outerIterations = outerIterations
    self.dataset = dataset
    self.clusters = []
    self.centroids = []

  def iteratingTheAlgorithm(self):
    for i in range(self.outerIterations):
      clusters, centroids = self.algorithm()
      self.clusters.append(clusters)
      self.centroids.append(centroids)

  def __repr__(self):
    representationString = f'Using k = {self.k} :-\n'
    for i in range(len(self.clusters)):
      representationString += f'\n\tTrial {i+1}:\n'
      for j in range(self.k):
        representationString += f'\t\tC{j+1}: {set(self.clusters[i][j])}\n'
    representationString += f'\nFinal Centroids: {self.centroids[len(self.centroids)-1]}\n'
    return representationString

  def findCentroids(self, clusters):
    centroids = []
    for i in clusters:
      xSum, ySum = 0, 0
      for j in i:
        xSum += j[0]
        ySum += j[1]
      centroids.append((xSum/len(i), ySum/len(i)))
    return centroids

  def algorithm(self):
    initialCentroids = random.sample(self.dataset, self.k)
    previousCentroids = initialCentroids
    previousClusters = [[] for i in range(self.k)]
    for i in range(self.innerIterations):
      if i == 0:
        for j in self.dataset:
          distOfDataPointFromEachCentroid = [(idx ,((j[0] - k[0])**2 + (j[1] - k[1])**2)**(0.5)) for idx, k in enumerate(initialC
          centroidNearestToDataPoint = sorted(distOfDataPointFromEachCentroid, key=lambda x: x[1])[0][0]
          previousClusters[centroidNearestToDataPoint].append(j)
        tmp = []
        for j in previousClusters:
          tmp.append(frozenset(j))
        previousClusters = tmp
      else:
        currentCentroids = self.findCentroids(previousClusters)
        if set(currentCentroids) == set(previousCentroids):
          return previousClusters, previousCentroids
        else:
          previousCentroids = currentCentroids
          currentClusters = [[] for i in range(self.k)]
          for j in self.dataset:
            distOfDataPointFromEachCentroid = [(idx ,((j[0] - k[0])**2 + (j[1] - k[1])**2)**(0.5)) for idx, k in enumerate(previo
            centroidNearestToDataPoint = sorted(distOfDataPointFromEachCentroid, key=lambda x: x[1])[0][0]
            currentClusters[centroidNearestToDataPoint].append(j)
          tmp = []
          for j in currentClusters:
            tmp.append(frozenset(j))
          currentClusters = tmp
          if set(currentClusters) == set(previousClusters):
            return previousClusters, previousCentroids
          else:
            previousClusters = currentClusters
    return previousClusters, previousCentroids
```

```
# Input For K and Iteration and Plotting the graph
print(f'Dataset name: Random 2-D dataset\n', f'\tx = {x}', f'\ty = {y}', sep='\n')

while(True):

  k = int(input('\nEnter the value of k: '))
  print(f'The value of k is: {k}\n')

  innerIterations = int(input('Number of iterations in the algorithm: '))
  print(f'Number of iterations in the algorithm are: {innerIterations}\n')

  outerIterations = int(input('Number of times the algorithm should run: '))
  print(f'Number of times the algorithm should run is: {outerIterations}\n')

  if k > len(dataset):
    print(f'Error: 0 < Value of k <= number of points in dataset ({len(x)})\n')
  else:
    kMeansClustering = KMeansClustering(dataset, k, innerIterations, outerIterations)
    kMeansClustering.iteratingTheAlgorithm()
    print(kMeansClustering)

  if k == 2:
    clusters = kMeansClustering.clusters[len(kMeansClustering.centroids)-1]
    xCluster1 = []
    yCluster1 = []
    xCluster2 = []
    yCluster2 = []

    for i in range(len(clusters)):
      tmp = list(clusters[i])

      if i == 0:
        for j in clusters[i]:
          xCluster1.append(j[0])
          yCluster1.append(j[1])

      if i == 1:
        for j in clusters[i]:
          xCluster2.append(j[0])
          yCluster2.append(j[1])

    plt.scatter(xCluster1, yCluster1, color='deepskyblue')
    plt.scatter(xCluster2, yCluster2, color='limegreen')
    plt.scatter(kMeansClustering.centroids[len(kMeansClustering.centroids)-1][0][0],
                kMeansClustering.centroids[len(kMeansClustering.centroids)-1][0][1],
                color='black')
    plt.scatter(kMeansClustering.centroids[len(kMeansClustering.centroids)-1][1][0],
                kMeansClustering.centroids[len(kMeansClustering.centroids)-1][1][1],
                color='black')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'{k}- Means Clusters')
    plt.show()

  elif k == 3:
    clusters = kMeansClustering.clusters[len(kMeansClustering.centroids)-1]
    xCluster1 = []
    yCluster1 = []
    xCluster2 = []
    yCluster2 = []
    xCluster3 = []
    yCluster3 = []

    for i in range(len(clusters)):
      tmp = list(clusters[i])
      if i == 0:
        for j in clusters[i]:
          xCluster1.append(j[0])
          yCluster1.append(j[1])
      if i == 1:
        for j in clusters[i]:
          xCluster2.append(j[0])
          yCluster2.append(j[1])
      if i == 2:
        for j in clusters[i]:
          xCluster3.append(j[0])
          yCluster3.append(j[1])

    plt.scatter(xCluster1,yCluster1, color='deepskyblue')
    plt.scatter(xCluster2,yCluster2, color='limegreen')
    plt.scatter(xCluster3,yCluster3, color='coral')
    plt.scatter(kMeansClustering.centroids[len(kMeansClustering.centroids)-1][0][0],
                kMeansClustering.centroids[len(kMeansClustering.centroids)-1][0][1],
                color='black')
    plt.scatter(kMeansClustering.centroids[len(kMeansClustering.centroids)-1][1][0],
                kMeansClustering.centroids[len(kMeansClustering.centroids)-1][1][1],
                color='black')
    plt.scatter(kMeansClustering.centroids[len(kMeansClustering.centroids)-1][2][0],
                kMeansClustering.centroids[len(kMeansClustering.centroids)-1][2][1],
                color='black')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'{k}- Means Clusters')
    plt.show()

  if k > 3 and k <= len(x):
    plt.scatter(x, y, color='deepskyblue')
    for i in range(k):
      plt.scatter(kMeansClustering.centroids[len(kMeansClustering.centroids)-1][i][0],
                  kMeansClustering.centroids[len(kMeansClustering.centroids)-1][i][1],
                  color='black')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'{k}- Means Clusters')
    plt.show()

  toContinue = True

  while(True):
    choice = input('Do you want to continue? (Y/N): ')

    if choice == 'Y':
      toContinue = True
      break
    elif choice == 'N':
      toContinue = False
      break
    else:
      print('INVALID CHOICE!\n')

  if toContinue:
    continue
  else:
    break
```

```
Dataset name: Random 2-D dataset

        x = [30, 23, 21, 26, 51, 20, 71, 52, 89, 15, 67, 72, 42, 44, 58, 83, 4, 16, 47, 55, 8, 94, 80, 49, 54, 53, 0, 78, 7
        y = [72, 90, 57, 28, 89, 93, 98, 59, 8, 95, 99, 77, 66, 47, 51, 41, 4, 36, 80, 92, 78, 42, 96, 87, 40, 31, 13, 34,

Enter the value of k: 2
The value of k is: 2

Number of iterations in the algorithm: 300
Number of iterations in the algorithm are: 300

Number of times the algorithm should run: 2
Number of times the algorithm should run is: 2

Using k = 2 :-

        Trial 1:
                C1: {(21, 57), (32, 54), (67, 99), (24, 70), (26, 28), (43, 86), (37, 56), (51, 89), (49, 87), (29, 35), (5
                C2: {(84, 58), (70, 73), (72, 77), (98, 9), (93, 30), (81, 62), (76, 49), (59, 63), (65, 75), (99, 68), (82

        Trial 2:
                C1: {(98, 9), (60, 7), (93, 30), (26, 28), (76, 49), (29, 35), (36, 43), (82, 3), (16, 36), (96, 25), (1, 1
                C2: {(21, 57), (70, 73), (32, 54), (72, 77), (67, 99), (84, 58), (24, 70), (81, 62), (43, 86), (37, 56), (5

Final Centroids: [(54.58, 24.68), (44.42, 74.32)]
```



```
Do you want to continue? (Y/N): Y

Enter the value of k: 3
The value of k is: 3

Number of iterations in the algorithm: 300
Number of iterations in the algorithm are: 300

Number of times the algorithm should run: 2
Number of times the algorithm should run is: 2

Using k = 3 :-

        Trial 1:
                C1: {(21, 57), (70, 73), (32, 54), (72, 77), (67, 99), (24, 70), (43, 86), (37, 56), (51, 89), (49, 87), (5
                C2: {(48, 17), (5, 48), (38, 38), (26, 28), (14, 46), (29, 35), (11, 50), (33, 1), (36, 43), (22, 0), (6, 5
                C3: {(84, 58), (98, 9), (93, 30), (81, 62), (76, 49), (99, 68), (82, 3), (96, 25), (85, 29), (90, 10), (66,

        Trial 2:
                C1: {(98, 9), (60, 7), (93, 30), (82, 3), (96, 25), (85, 29), (90, 10), (66, 15), (45, 21), (75, 6), (48, 1
                C2: {(84, 58), (70, 73), (63, 65), (72, 77), (67, 99), (55, 92), (79, 67), (81, 62), (43, 86), (73, 74), (5
                C3: {(21, 57), (32, 54), (24, 70), (26, 28), (37, 56), (29, 35), (14, 46), (36, 43), (16, 36), (1, 16), (4,

Final Centroids: [(70.71875, 20.3125), (65.3103448275862, 74.75862068965517), (20.333333333333332, 54.666666666666664)]
```



```
Do you want to continue? (Y/N): N
```

```
        Do you want to continue? (Y/N): N
```

✓ 44s completed at 22:46