**Practical 6:**

**Aim: Implement the k-means clustering algorithm.**

**Solution:**

**Code: Creating dataset for kmeans**

import numpy as np

import math

import pandas as pd

import matplotlib.pyplot as plt

import random

np.random.seed(1)

def circulo(num\_datos=100, R=1, minimo=0, maximo=1, center\_x=0, center\_y=0):

    pi = math.pi

    r = R \* np.sqrt(np.random.uniform(minimo, maximo, size=num\_datos))

    theta = np.random.uniform(minimo, maximo, size=num\_datos) \* 2 \* pi

    x = center\_x + np.cos(theta) \* r

    y = center\_y + np.sin(theta) \* r

    x = np.round(x, 3)

    y = np.round(y, 3)

    df = np.column\_stack([x, y])

    df = pd.DataFrame(df)

    df.columns = ['x', 'y']

    return(df)

# Create data

datos\_1 = circulo(num\_datos=20, R=10, center\_x=5, center\_y=30)

datos\_2 = circulo(num\_datos=20, R=10, center\_x=20, center\_y=10)

datos\_3 = circulo(num\_datos=20, R=10, center\_x=50, center\_y=50)

data = pd.concat([datos\_1, datos\_2, datos\_3], axis=0)

data.head()

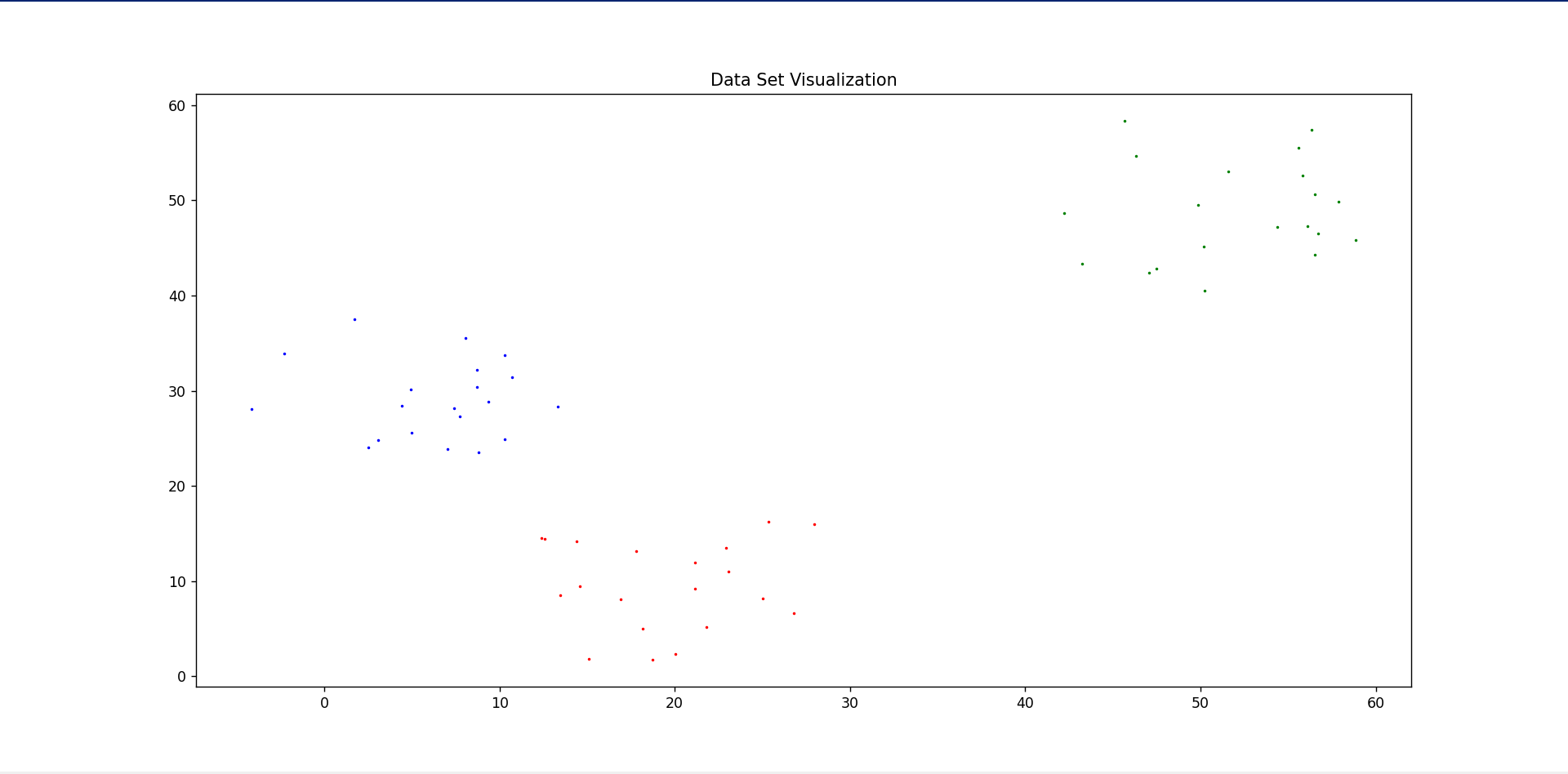
plt.scatter(datos\_1['x'], datos\_1['y'], c='b', s=1.0)

plt.scatter(datos\_2['x'], datos\_2['y'], c='r', s=1.0)

plt.scatter(datos\_3['x'], datos\_3['y'], c='g', s=1.0)

plt.title('Data Set Visualization')

**Output**

****

**Code:**

**2. Implementation of kmeans from scratch**

# implements kmeans from scratch

def kmeans(df, k, data\_num):

    # chooses different random points from dataset for initial clusters

    x\_list, y\_list = generate\_random\_points(df, k)

    df\_copy = df.copy()  # copying data set

    check = True

    objective\_function = []  # stores all object function values in each iteration

    iter = 0  # number of iteration

    points\_colors = {}  # colors for the clusters

    while (check):

        i = 0

        # calculates distance for every point according to its cluster centroid

        df\_copy, total\_points, obj\_func = calculate\_dist(

            df\_copy, x\_list, y\_list, k)

        # sum of objective function values

        total\_obf = 0

        for obj in obj\_func.values():

            total\_obf += obj

        # adds current objective function value to the list

        objective\_function.append(total\_obf)

        # stops the while loop when objective function starts to converge

        if abs(objective\_function[iter - 1] - objective\_function[iter]) < 0.01 and iter >= 1:

            check = False

        # calculates new values for centroids

        for coordinates in total\_points.values():

            new\_x, new\_y = calculate\_mean(coordinates)

            x\_list[i] = new\_x

            y\_list[i] = new\_y

            i += 1

        # plots the initial cluster centers

        if iter == 0:

            points\_colors = initial\_cluster\_centers\_plot(

                df\_copy, x\_list, y\_list, k, data\_num)

        iter += 1

    final\_cluster\_centers\_plot(

        df\_copy, x\_list, y\_list, points\_colors, iter, objective\_function[-1])  # plots final

# calculates euclid distance for two points

def euclid\_calculator(x1, y1, x2, y2):

    # euclid distance formula

    distance = np.sqrt(np.square(abs(x1 - x2)) + np.square(abs(y1 - y2)))

    return distance

def calculate\_dist(df\_dist, x\_list, y\_list, k):

    obj\_func = {}  # stores object functions for each distances

    total\_points = {}  # stores all data points for each cluster centroids

    # creates points with names according to number of k

    for x in range(1, k + 1):

        name = 'point\_'

        obj\_func[name + str(x)] = 0

        total\_points[name + str(x)] = []

    for index, row in df\_dist.iterrows():

        points\_distance = {}

        for x in range(1, k+1):

            name = 'point\_'

            points\_distance[name + str(x)] = 0

        for x, y, point in zip(x\_list, y\_list, points\_distance.keys()):

            # calculates distance for determine point is close to which center

            distance = euclid\_calculator(x, y, row['x'], row['y'])

            # stores points with distances

            df\_dist.loc[index, point] = distance

            points\_distance[point] = distance

        # sorts for finding every point close to which center

        sorted\_dist = sorted(points\_distance.items(), key=lambda kv: kv[1])

        point = [row['x'], row['y']]

        point\_type = sorted\_dist[0][0]

        obj\_func[point\_type] += np.square(sorted\_dist[0][1])

        df\_dist.loc[index, 'point\_type'] = point\_type

        total\_points[point\_type].append(tuple(point))

    return df\_dist, total\_points, obj\_func

# calculates min x and y values for finding the new centroids

def calculate\_mean(dist\_list):

    x\_total = 0

    y\_total = 0

    for item in dist\_list:

        x\_total += item[0]

        y\_total += item[1]

    mean\_x = x\_total / len(dist\_list)

    mean\_y = y\_total / len(dist\_list)

    return mean\_x, mean\_y

# calculates sum of all distances

def total\_dist(dist):

    dist\_sum = 0

    for item in dist:

        dist\_sum += item

    return dist\_sum

# visualizing scatter plot version of data

def scatter\_plot(plotted\_df):

    fig, ax = plt.subplots()

    # selecting x and y axises without labels

    ax.scatter(plotted\_df['x'], plotted\_df['y'], c=plotted\_df['class'])

    plt.xlabel('x')  # entering name for x axis

    plt.ylabel('y')  # entering name for y axis

    plt.title("Dataset")  # entering a title for plot

    plt.show()

# plots the initial cluster centers

def initial\_cluster\_centers\_plot(initial\_df, x\_list, y\_list, k, data\_num):

    fig, ax = plt.subplots()

    # defining colors for each clustered points at the beginning according to k number

    points\_colors = {'point\_1': 'b', 'point\_2': 'r', 'point\_3': 'g', 'point\_4': 'y',

                     'point\_5': 'c', 'point\_6': 'm', 'point\_7': 'k', 'point\_8': 'w'}

    # selecting x and y axises with point type of each row

    ax.scatter(initial\_df['x'], initial\_df['y'],

               c=initial\_df['point\_type'].map(points\_colors), s=1.0)

    # draws the new centroids according to mean values

    for x, y in zip(x\_list, y\_list):

        plt.scatter(x, y, c='black', s=10)

    plt.xlabel('x \n Initial Cluster Centers \n Dataset=' +

               str(data\_num)+'\n k='+str(k))

    plt.ylabel('y')

    plt.title("k-Means")

    plt.show()

    return points\_colors

# plots the final cluster centers

def final\_cluster\_centers\_plot(final\_df, x\_list, y\_list, points\_colors, iter\_num, final\_obf):

    fig, ax = plt.subplots()

    ax.scatter(final\_df['x'], final\_df['y'],

               c=final\_df['point\_type'].map(points\_colors), s=1.0)

    for x, y in zip(x\_list, y\_list):

        plt.scatter(x, y, c='black', s=10)

    plt.xlabel('Final Cluster Centers\n Iteration Count=' +

               str(iter\_num) + '\n Objective Function Value:' + str(final\_obf))

    plt.ylabel('y')

    plt.title("k-Means")

    plt.show()

# generates random points for initial cluster centroids

def generate\_random\_points(dataset, k):

    rand\_x, rand\_y = [], []

    random\_points = random.sample(range(1, len(dataset) - 1), k)

    for rand\_index in random\_points:

        rand\_x.append(dataset.iloc[rand\_index][0])

        rand\_y.append(dataset.iloc[rand\_index][1])

    return rand\_x, rand\_y

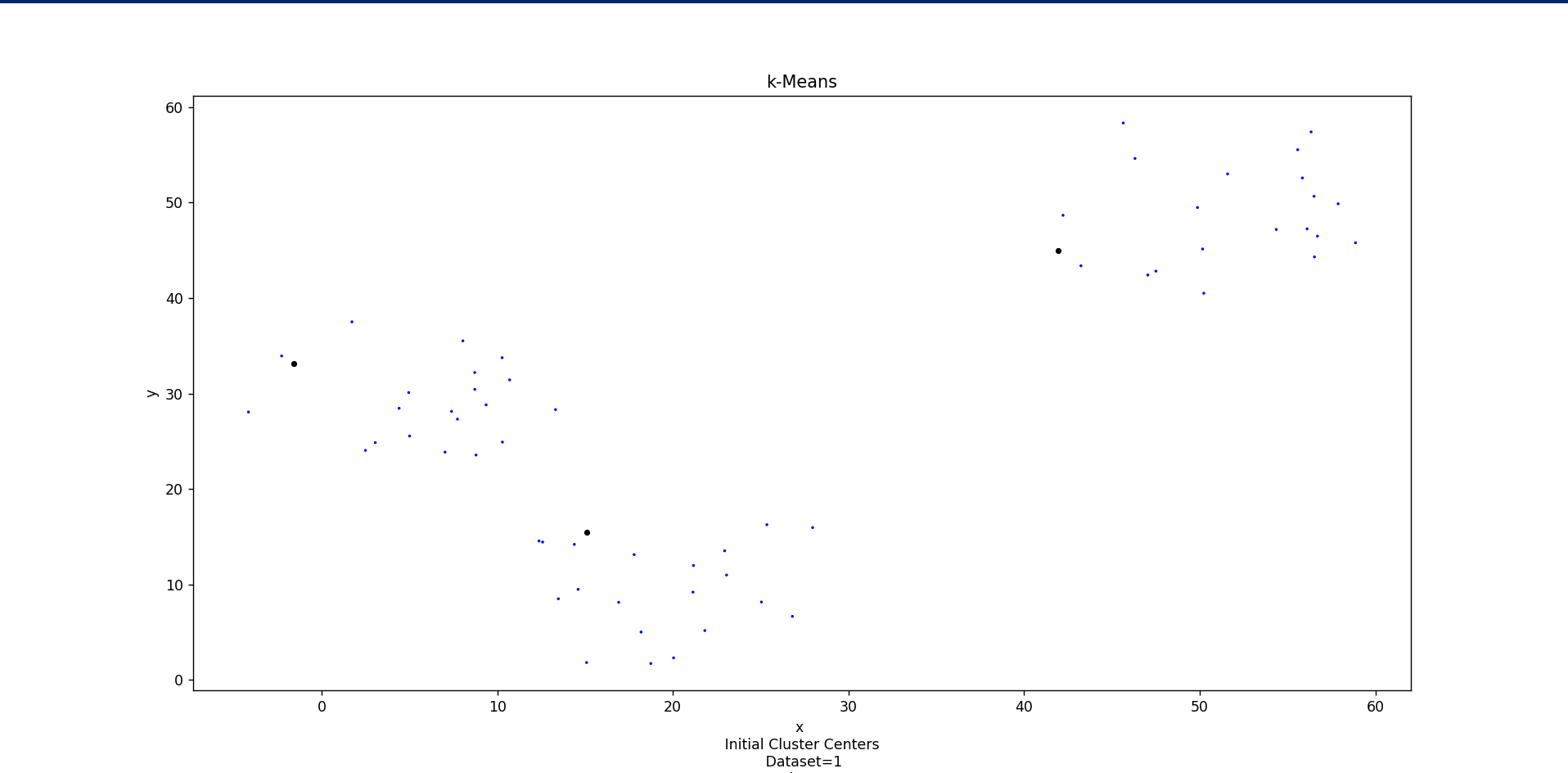
k = 3

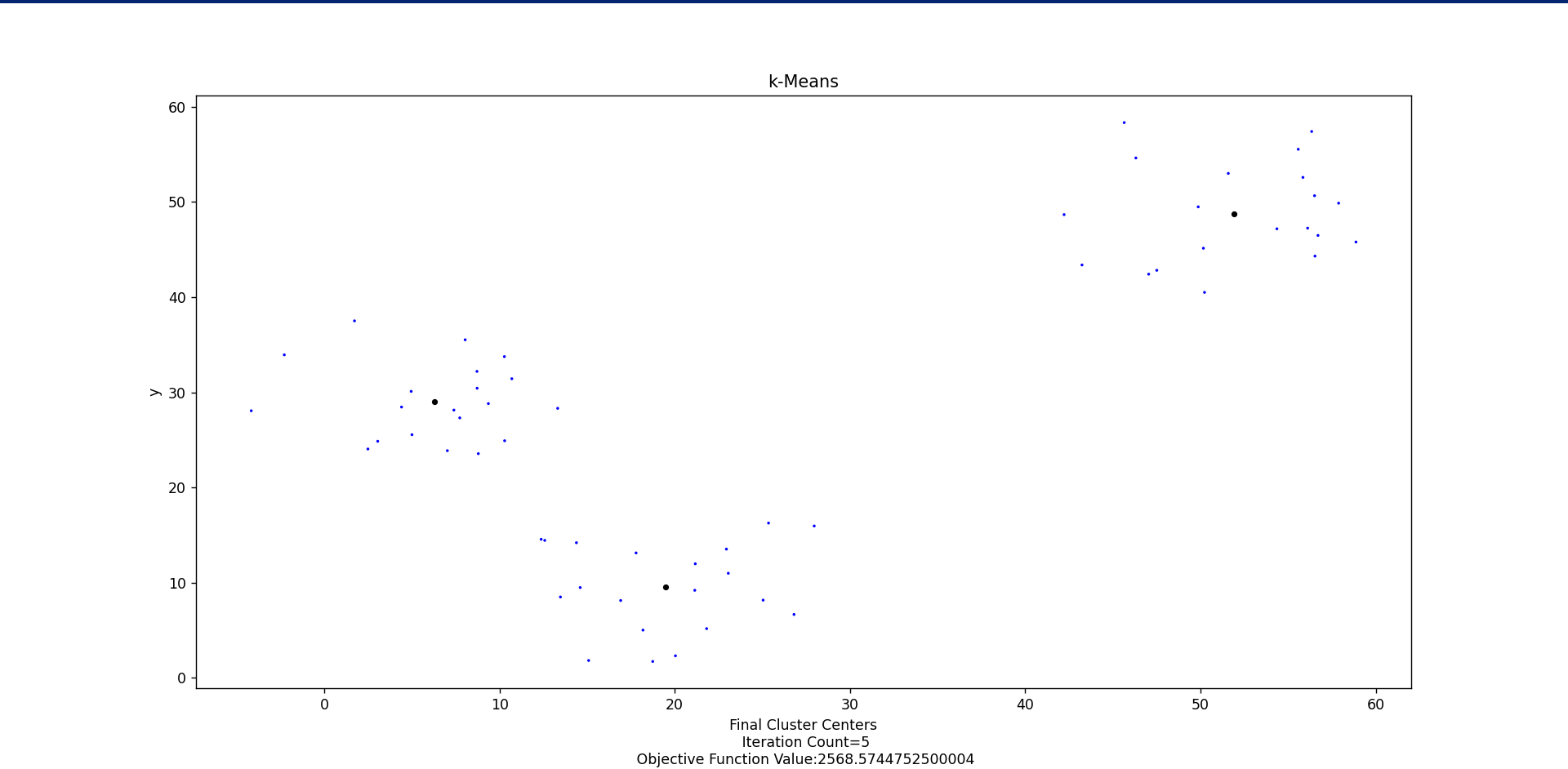
data\_num = 1

# Create object of KMeans class

km = kmeans(data, k, data\_num)

**Output:**

****

****