Abby Ortego W0716476

Data Generation & Clustering

Imports

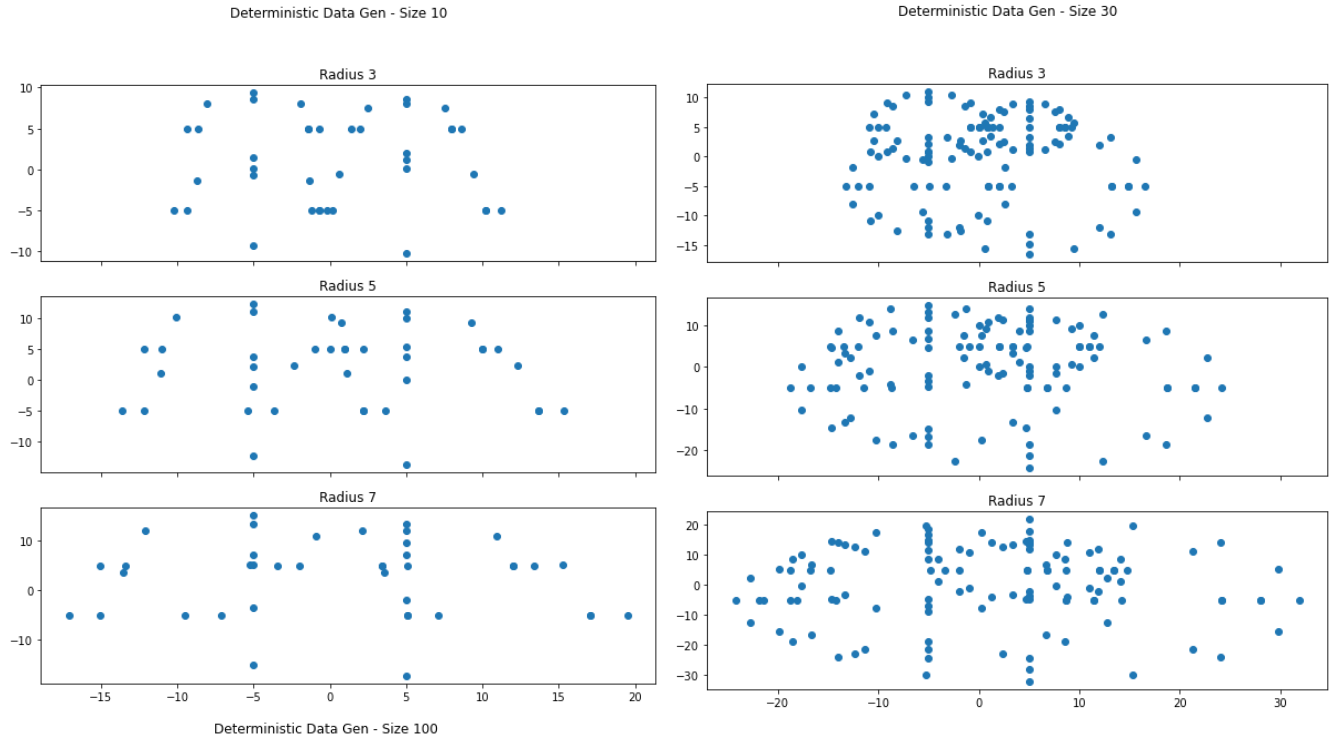
```
import matplotlib.pyplot as plt
import numpy as np
import random as r
import math as m
```

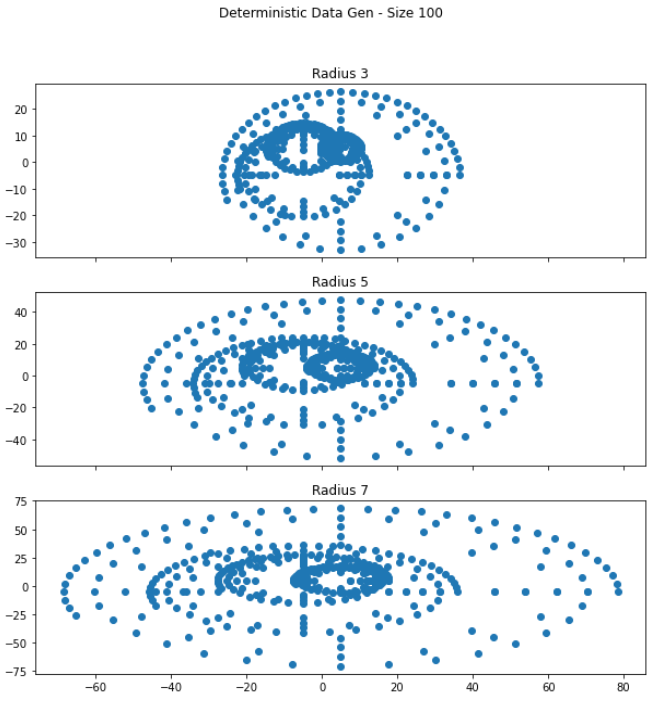
Data Generation

Deterministic Data Generation

```
def determDataGen(numClusters, centX, centY, clustSize, clustRad):
    # cluster x and y coordinates
    px = []
    py = []
    for j in range(numClusters):
        # cluster centers
        cx = centX[j]
        cy = centY[j]
        ang = 0
        radInc = clustRad / 5
        pointsPerRad = 4
        angInc = 360 / pointsPerRad
        for k in range(clustSize):
            myX = cx + clustRad * np.cos(np.deg2rad(ang))
            myY = cy + clustRad * np.sin(np.deg2rad(ang))

            # append new cluster point
            px.append(myX)
            py.append(myY)

            ang = ang + angInc
            if(ang > 360):
                ang = 0
                clustRad += radInc
                pointsPerRad *= 2
```

```
                angInc = 360 / pointsPerRad
            #
        #
    #
    return px, py
# determDataGen
```



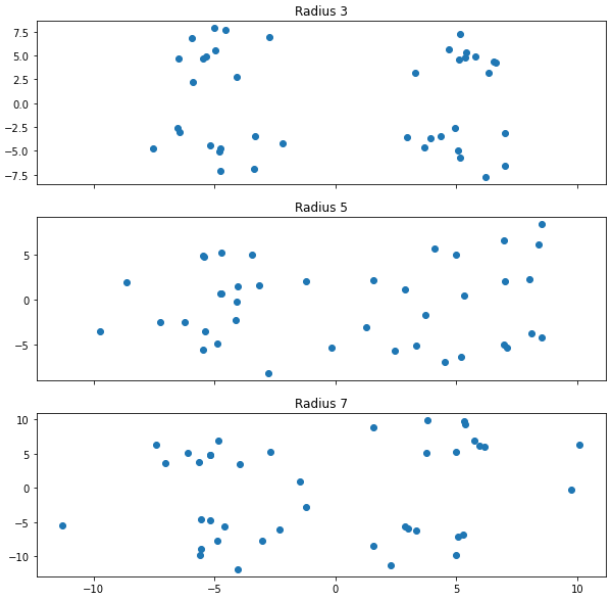


Non-Deterministic Data Generation

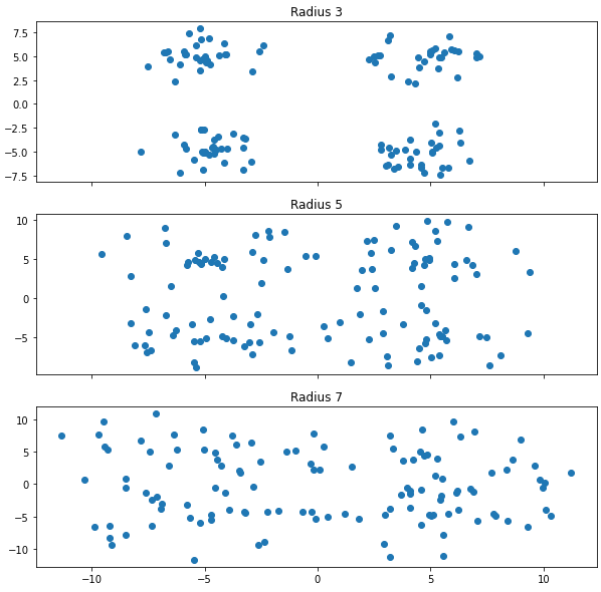
```
def nonDetermDataGen(numClusters,
centX, centY, clustSize, clustRad):
    # cluster x and y coordinates
    px = []
    py = []
    for clust in range(numClusters):
        # cluster centers
        cx = centX[clust]
        cy = centY[clust]

        for clust_pnt in
range(clustSize):
            angD = r.randint(0,359)
            magnitude = r.random() *
clustRad
```

```
            myX = cx + magnitude * np.cos(np.deg2rad(angD))
            myY = cy + magnitude * np.sin(np.deg2rad(angD))

            # append new cluster point
            px.append(myX)
            py.append(myY)
        #
    #
    return px, py
# nonDetermDataGen
```

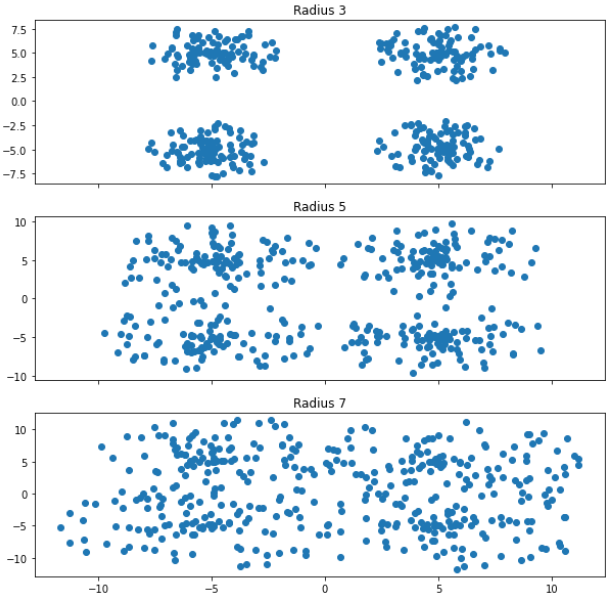
Non Deterministic Data Gen - Size 10



Non Deterministic Data Gen - Size 30



Non Deterministic Data Gen - Size 100



Function Calls to Generate Plots:

```
pop_size = [10, 30, 100]
rad_size = [3, 5, 7]

determData = {}
nonDetermData = {}
for ind_pop, pop in enumerate(pop_size):

    det_fig, det_axs = plt.subplots(len(pop_size), sharex = True, figsize = (10,
10))
    det_fig.suptitle(f'Deterministic Data Gen - Size {pop}')

    nonDet_fig, nonDet_axs = plt.subplots(len(pop_size), sharex = True, figsize =
(10, 10))
    nonDet_fig.suptitle(f'Non Deterministic Data Gen - Size {pop}')

    for ind_rad, rad in enumerate(rad_size):
        key = f's{pop}_r{rad}'

        det_data_gen = determDataGen(4, [5, -5, -5, 5], [5, 5, -5, -5], pop, rad)
        det_axs[ind_rad].scatter(det_data_gen[0], det_data_gen[1])
        det_axs[ind_rad].set_title(f'Radius {rad}')

        determData[key] = det_data_gen

        nonDet_data_gen = nonDetermDataGen(4, [5, -5, -5, 5], [5, 5, -5, -5],
pop, rad)
        nonDet_axs[ind_rad].scatter(nonDet_data_gen[0], nonDet_data_gen[1])
        nonDet_axs[ind_rad].set_title(f'Radius {rad}')

        nonDetermData[key] = nonDet_data_gen
    #
#
```

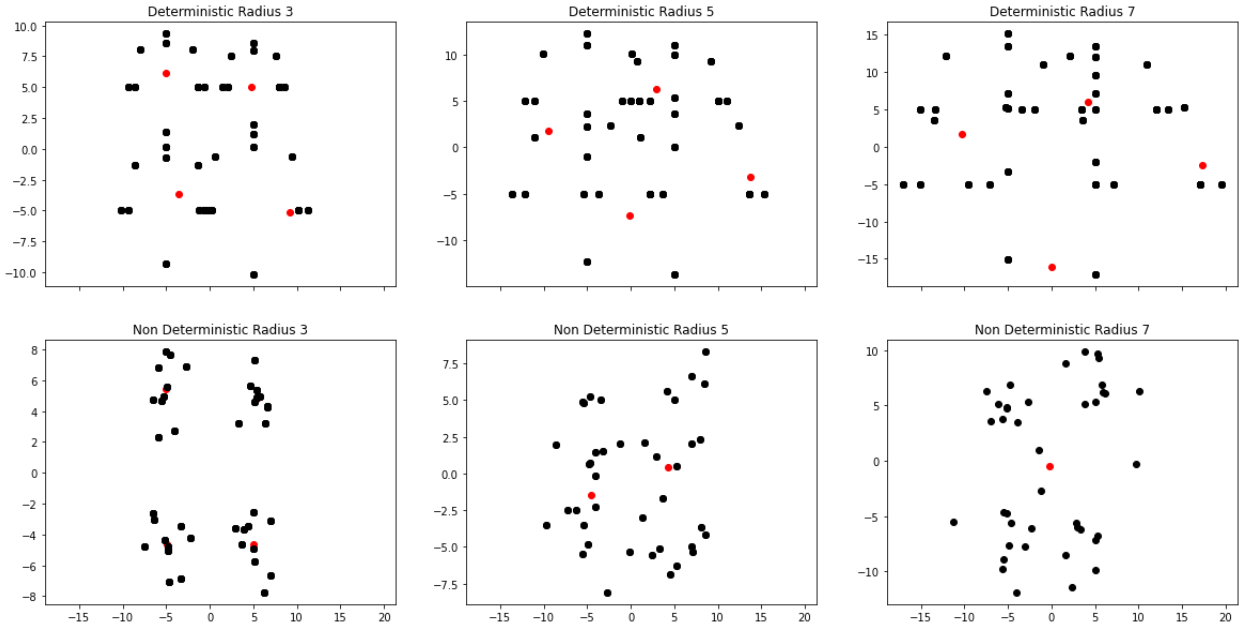
Clustering

Ad Hoc Clustering

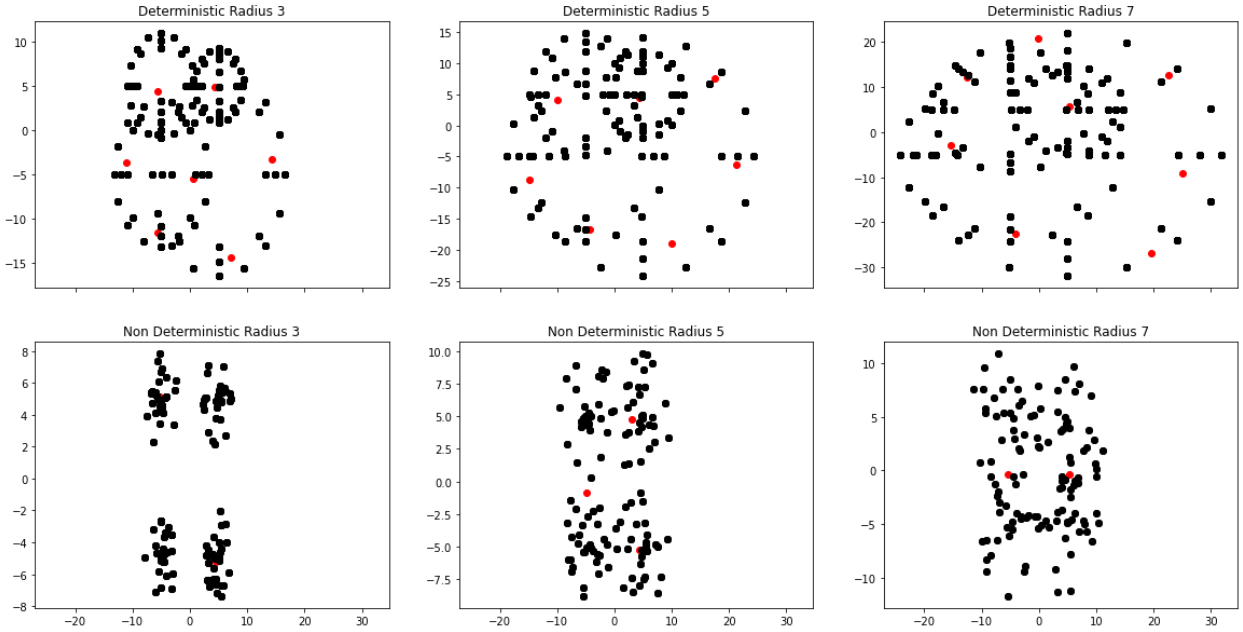
```
def adHocClustering(x, y, maxDist):
    nClusts = 0 # number of clusters
    clusters = [] # the clusters
```

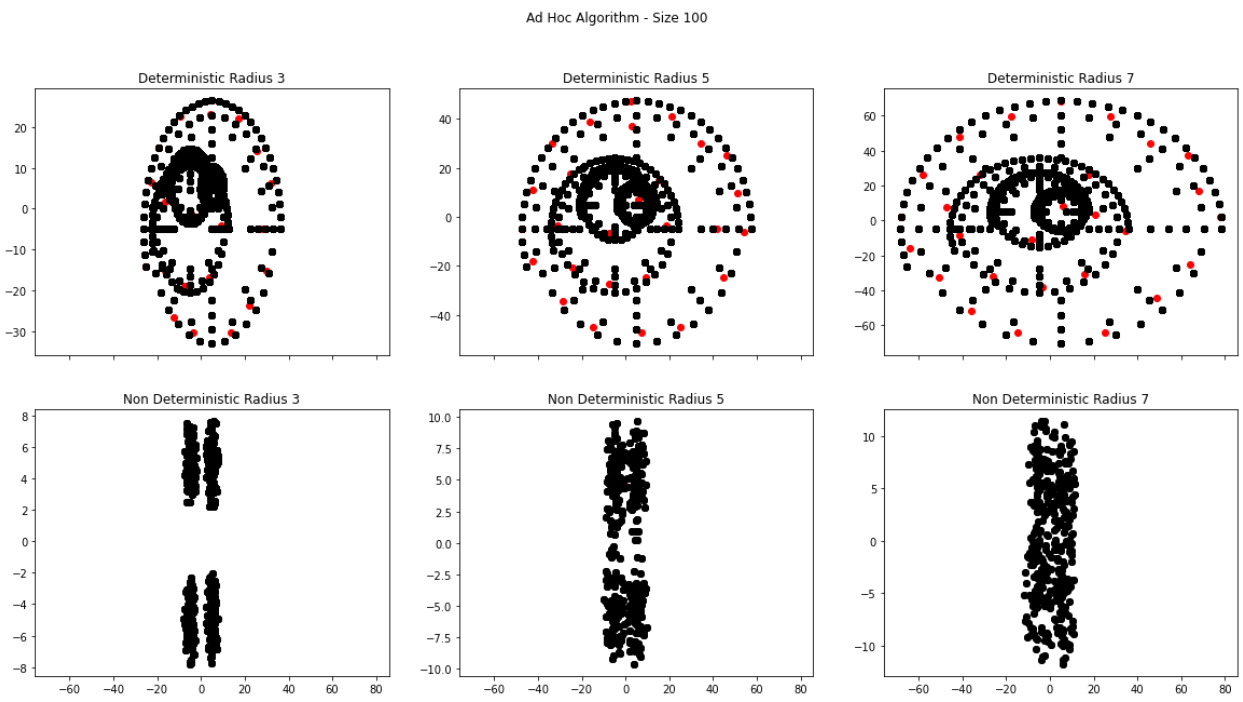
```
    pntsPerCluster = [] # number of pnts per cluster

    for j in range(len(x)): # loop through each data pnt
        pnt = [x[j], y[j]] # for convenience

        if(nClusts == 0): # no clusters?
            clusters.append(pnt) # this is the first pnt of the first cluster
            nClusts = nClusts + 1 # increment clusters count and...
            pntsPerCluster.append(1) # create item to increment pnts for that
cluster

        else: # find closest cluster
            dists = [m.dist(pnt, cluster) for cluster in clusters]
            closeDists = min(dists) # closest cluster
            closeIndex = dists.index(min(dists)) # index of the closest cluster

            if(closeDists < maxDist): # close enough
                # most and rest counts for merging
                most = pntsPerCluster[closeIndex] / (pntsPerCluster[closeIndex] +
1)
                rest = 1 / (pntsPerCluster[closeIndex] + 1)

                # merge centroid
                clusters[closeIndex] = [((clusters[closeIndex][0] * most) +
(pnt[0] * rest)), ((clusters[closeIndex][1] * most) + (pnt[1] * rest))]
                pntsPerCluster[closeIndex] = pntsPerCluster[closeIndex] + 1 #
increment pntsPerCluster

            else: # new cluster
                clusters.append(pnt) # first pnt of a new cluster
                nClusts = nClusts + 1 # increment clusters and...
                pntsPerCluster.append(1) # add new item to increment points for
that cluster
            #
        #
    #

    return nClusts, clusters, pntsPerCluster

# adHocClustering
```

Ad Hoc Algorithm - Size 10



Ad Hoc Algorithm - Size 30





Function Calls to Generate Plots:

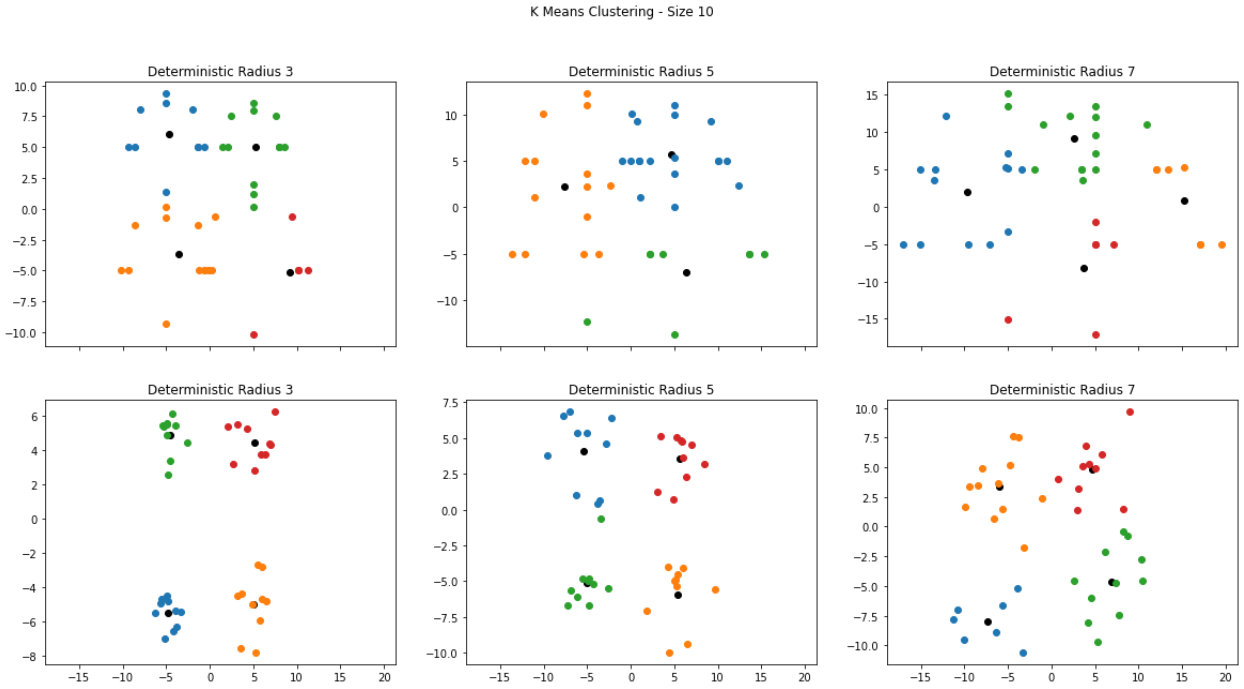
```
for ind_pop, pop in enumerate(pop_size):

    fig_10, axs_10 = plt.subplots(2, 3, sharex = True, figsize = (20, 10))
    fig_10.suptitle(f'Ad Hoc Algorithm - Size {pop}')

    max_dist = 8

    for ind_rad, rad in enumerate(rad_size):

        nClusts_determ, clusters_determ, pntsPerCluster_determ =
adHocClustering(determData[f's{pop}_r{rad}'][0], determData[f's{pop}_r{rad}'][1],
max_dist)
        for j in range(nClusts_determ):
            axs_10[0][ind_rad].scatter(clusters_determ[j][0],
clusters_determ[j][1], color = 'r')
            axs_10[0][ind_rad].scatter(determData[f's{pop}_r{rad}'][0],
determData[f's{pop}_r{rad}'][1], color = 'k')
        #
        axs_10[0][ind_rad].set_title(f'Deterministic Radius {rad}')

        nClusts_nonDeterm, clusters_nonDeterm, pntsPerCluster_nonDeterm =
adHocClustering(nonDetermData[f's{pop}_r{rad}'][0],
nonDetermData[f's{pop}_r{rad}'][1], max_dist)
```

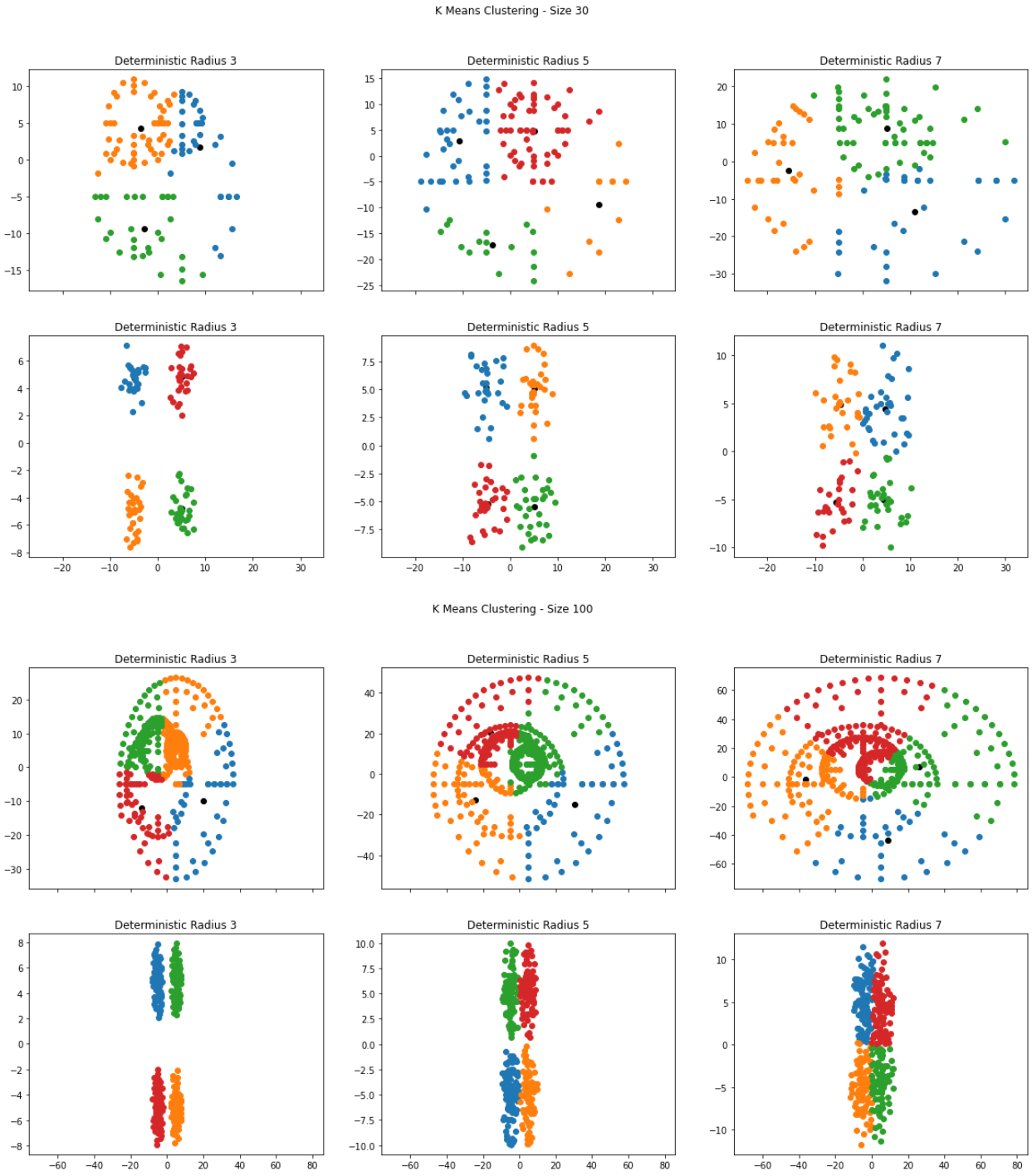
```
        for j in range(nClusts_nonDeterm):
            axs_10[1][ind_rad].scatter(clusters_nonDeterm[j][0],
clusters_nonDeterm[j][1], color = 'r')
            axs_10[1][ind_rad].scatter(nonDetermData[f's{pop}_r{rad}'][0],
nonDetermData[f's{pop}_r{rad}'][1], color = 'k')
        #
        axs_10[1][ind_rad].set_title(f'Non Deterministic Radius {rad}')

        max_dist += 4
    #

#
```

K Means Clustering

```
def kMeans(x, y, k):

    centroids_x = []
    centroids_y = []
    # choose k random points >> new centers
    for i in range(k):
        rndm_indx = r.randint(0, len(x)-1)
        centroids_x.append(x[rndm_indx])
        centroids_y.append(y[rndm_indx])
    #

    clusters_x = {}
    clusters_y = {}
    for i in range(100):
        clusters_x = {}
        clusters_y = {}
        # assign data to clusters
        for centroid_indx in range(len(centroids_x)):
            pnts_x = [] # holds the list of points that match the current cluster
            pnts_y = []

            # loop through the points
            for pnt_indx in range(len(x)):
                # find dist from centers
                dist = [m.dist([centroids_x[j], centroids_y[j]], [x[pnt_indx],
y[pnt_indx]]) for j in range(len(centroids_x))]

                # if the min(dist) is this centroid assign it to that cluster
                if centroid_indx == dist.index(min(dist)):
                    pnts_x.append(x[pnt_indx])
```

```
                    pnts_y.append(y[pnt_indx])
                #

                # add list of cluster pnts to the clusters dict
                clusters_x[centroid_indx] = pnts_x
                clusters_y[centroid_indx] = pnts_y
            #
        #

        for key in clusters_x:
            avg_x = np.mean(clusters_x[key])
            avg_y = np.mean(clusters_y[key])

            centroids_x[key] = avg_x
            centroids_y[key] = avg_y
        #
    #

    return centroids_x, centroids_y, clusters_x, clusters_y

#
```





Function Calls to Generate Plots:

```
for ind_pop, pop in enumerate(pop_size):

    fig, axs = plt.subplots(2, 3, sharex = True, figsize = (20, 10))
    fig.suptitle(f'K Means Clustering - Size {pop}')
```

```
    for ind_rad, rad in enumerate(rad_size):
        centersX_determ, centersY_determ, clustX_determ, clustY_determ =
kMeans(determData[f's{pop}_r{rad}'][0], determData[f's{pop}_r{rad}'][1], 4)
        axs[0][ind_rad].scatter(centersX_determ, centersY_determ, color = 'k')
        for j in range(len(clustX_determ)):
            axs[0][ind_rad].scatter(clustX_determ[j], clustY_determ[j])
        axs[0][ind_rad].set_title(f'Deterministic Radius {rad}')

        centersX_nonDeterm, centersY_nonDeterm, clustX_nonDeterm,
clustY_nonDeterm = kMeans(nonDetermData[f's{pop}_r{rad}'][0],
nonDetermData[f's{pop}_r{rad}'][1], 4)
        axs[1][ind_rad].scatter(centersX_nonDeterm, centersY_nonDeterm, color =
'k')
        for j in range(len(clustX_nonDeterm)):
            axs[1][ind_rad].scatter(clustX_nonDeterm[j], clustY_nonDeterm[j])
        axs[1][ind_rad].set_title(f'Deterministic Radius {rad}')
    #
#
```