تابع initializecenters:

def initializecenters(x, k):  
 centers = []  
 for i in range(k):  
 index = random.randint(0, len(x))  
 centers.append(x[index])  
 return centers

در این تابع، مراکز اولیه از بین داده های موجود به صورت تصادفی انتخاب می شوند. به این صورت که به تعداد خوشه ها (k) که از پیش باید تعیین شده باشد، مقادیر تصادفی در بازه ی Index ها تولید می شود.

تابع calculateDistance:

def calculateDistance(x, y):  
 distance = 0  
 for i in range(len(x)):  
 distance += (x[i] - y[i])\*\*2  
 return distance\*\*0.5

در این تابع فاصله ی بین دو نقطه محاسبه می شود، که برای فاصله از معیار فاصله ی اقلیدسی استفاده شده است.

تابع findClosestCenters:

def findClosestCenters(x, centers):  
 x\_index = []  
 for point in x:  
 distances = []  
 for center in centers:  
 distances.append(calculateDistance(point, center))  
 x\_index.append(np.argmin(distances))  
 return x\_index

در این تابع، فاصله ی هر نقطه از دیتاست تا همه ی مراکز خوشه ها محاسبه می شود (فاصله از طریق تابع بالا محاسبه می شود). لیست distances این مقادیر را برای هر نقطه از دیتاست نگه میدارد، به این صورت که در ایندکس صفرم از این لیست، فاصله ی داده تا خوشه ی صفرم نگه داری می شود. سپس هر نقطه به خوشه ای نسبت داده می شود که فاصله ی کمتری تا مرکز آن خوشه دارد. نحوه ی انتساب به این صورت است که لیست x\_index برای هر نقطه در ایندکس I ام از دیتاست یک عدد بین 0 تا k را نگه می دارد. اگر داده ی I دارای مقدار 3 در لیست x\_index باشد یعنی به خوشه ی سوم تعلق دارد.

تابع computeMeans:

def computeMeans(x, x\_index, k, size, dimension):  
 centers = []  
 for i in range(k):  
 indices = [j for j, cluster in enumerate(x\_index) if cluster == i] # get the indices of points in dataset belong to the same cluster  
 sum\_dimensions = [0] \* dimension # maintain sum of values of each dimension  
 for idx in indices:  
 for d in range(dimension):  
 sum\_dimensions[d] += x[idx][d]  
 mean = []  
 for el in sum\_dimensions:  
 mean.append(el / len(indices))  
 centers.append(mean)  
 return centers

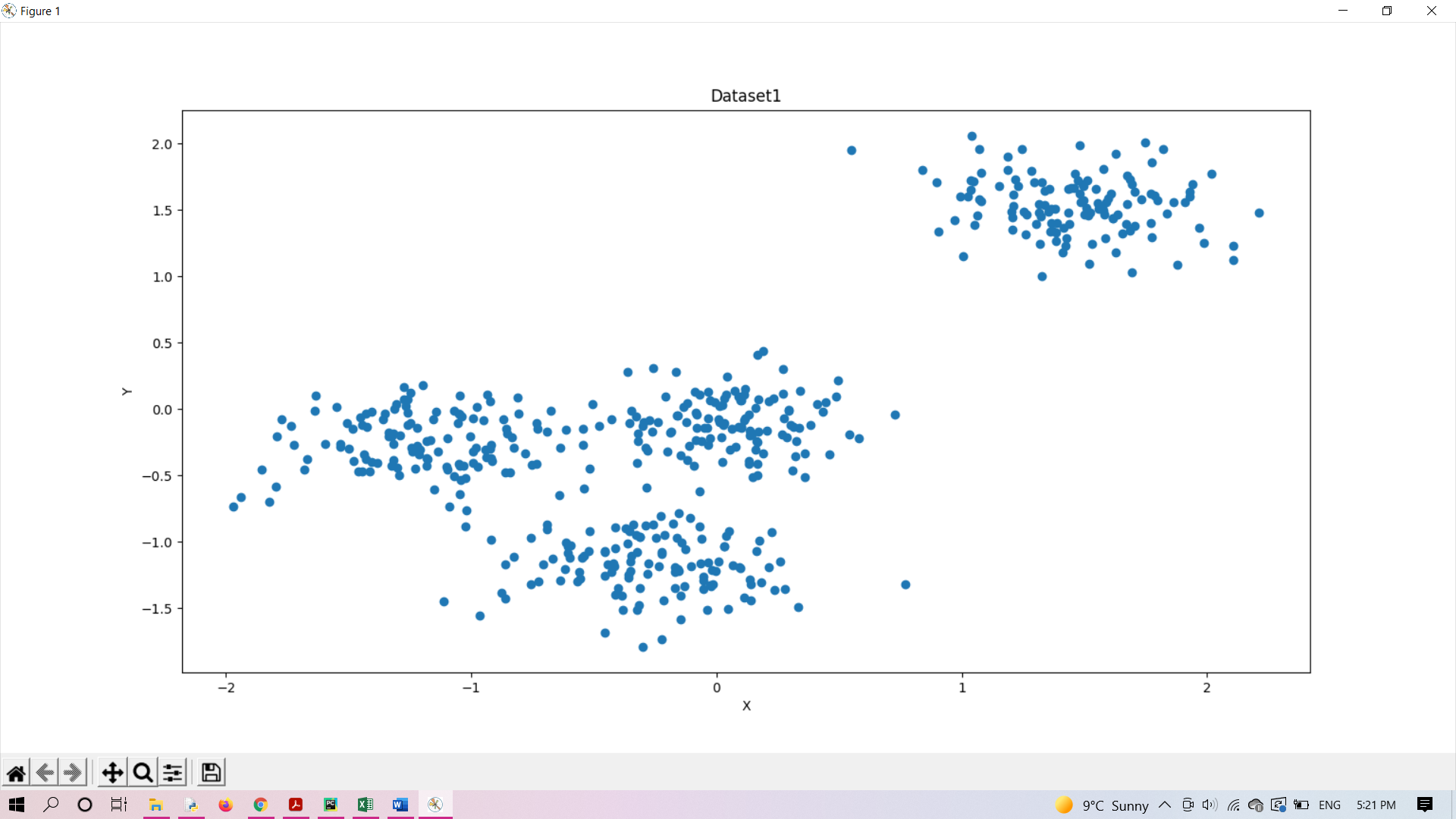
در این تابع، مراکز جدید بر حسب داده های هر خوشه محاسبه می شود. از آنجا که از معیار فاصله ی اقلیدسی در محاسبه ی فاصله ی نقاط استفاده شد، مرکز هر خوشه، میانگین داده های آن خوشه می باشد. برای این منظور، ایندکس داده هایی که دارای یک مقدار در x\_index هستند استخراج میشود. سپس برای هر بعد مقادیر داده ها با هم جمع شده و تقسیم بر تعداد داده های آن خوشه می شود. در نهایت مرکز جدید خوشه ی I ام به لیست مراکز اضافه می شود.

dataset = pd.read\_csv("Dataset1.csv")  
size, dimension = dataset.shape  
x = np.array(dataset.values)  
  
k = 4 # number of clusters  
centers = initializecenters(x, k)  
  
n = 15 # number of iterations  
for i in range(n):  
 x\_index = findClosestCenters(x, centers)  
 centers = computeMeans(x, x\_index, k, size, dimension)

در قطعه کد بالا، داده های Dataset1.csv خوانده شده است و اندازه و ابعاد آن ذخیره شده است (این کد به صورتی پیاده سازی شده است که برای ابعاد بالاتر داده ها هم قابل استفاده باشد). تعداد خوشه ها توسط پارامتر k تعیین می شود و به تعداد آن مراکز اولیه مقداردهی می شوند.

می توان تعداد تکرار الگوریتم را با n تغییر داد. در هر بار تکرار، داده ها به یک خوشه نسبت داده می شوند. بر اساس داده های جدید هر خوشه، مراکز جدید از طریق میانگین گیری بدست می آیند.

در تصویر زیر، دیتاست پیش از اعمال الگوریتم خوشه بندی k-means نمایش داده شده است.

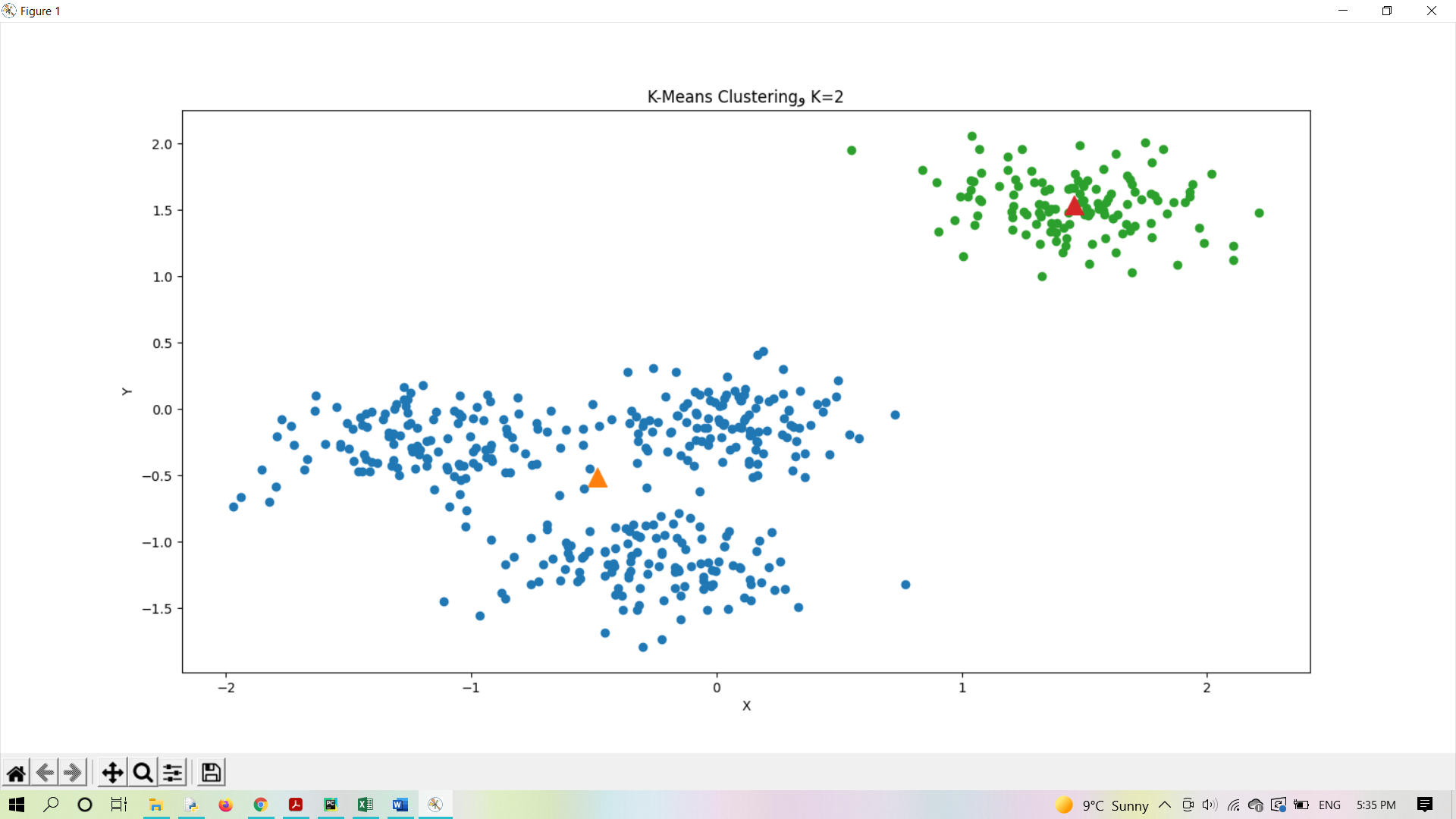


x\_dimension = []  
y\_dimension = []  
for i in range(size):  
 x\_dimension.append(x[i][0])  
 y\_dimension.append(x[i][1])  
plt.scatter(x\_dimension, y\_dimension)  
plt.title("Dataset1")  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.show()

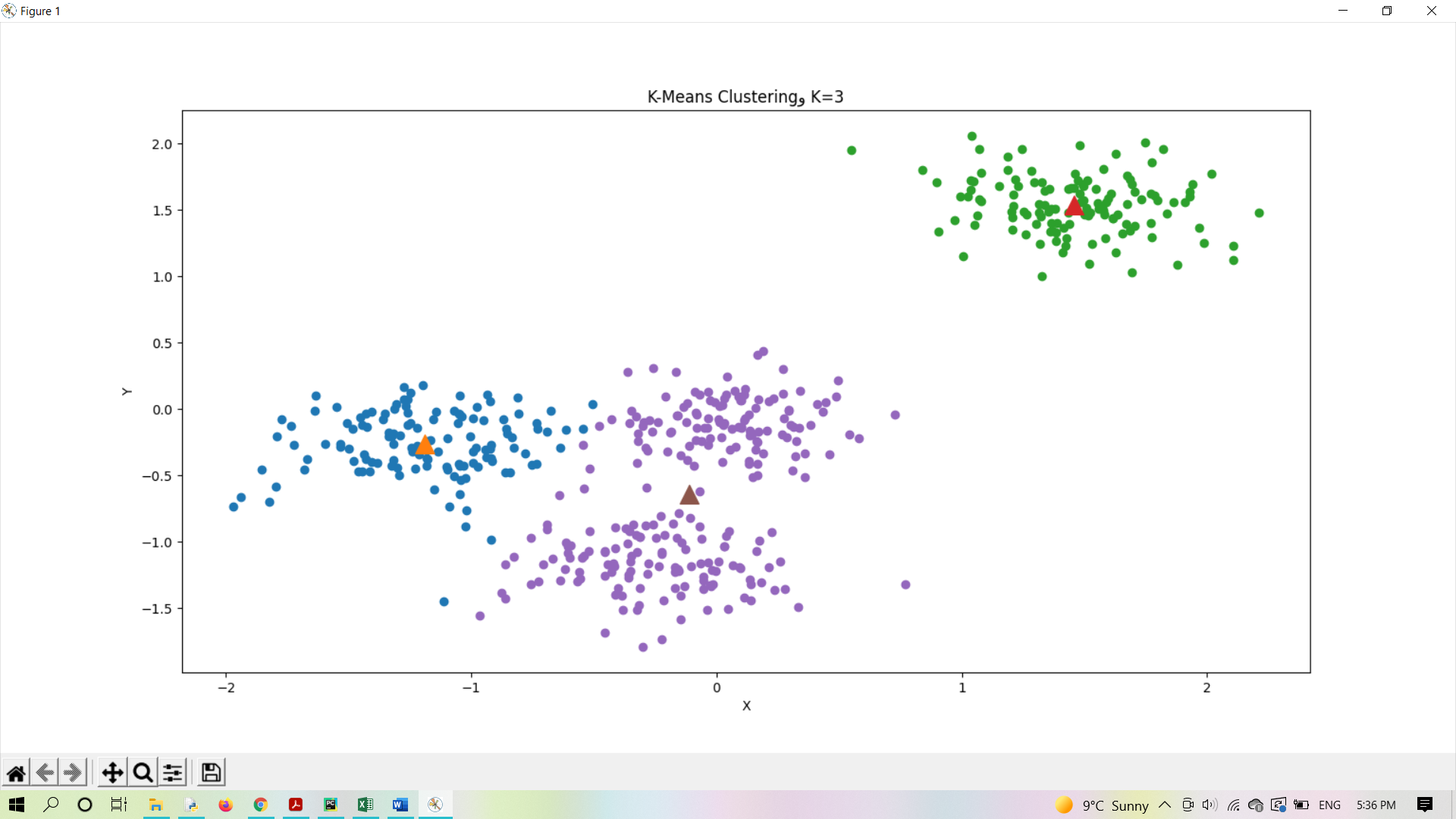
در قطعه کد بالا، دیتاست پیش از اعمال الگوریتم رسم شده است و در قطعه کد پایین پس از اعمال الگوریتم k-means، داده های هر خوشه با رنگ های مجزا رسم شده اند. علاوه بر آن مرکز هر خوشه با مثلث و رنگی متفاوت نمایش داده می شود.

for i in range(k):  
 indices = [j for j, cluster in enumerate(x\_index) if cluster == i] # get the indices of points in dataset belong to the same cluster  
 x\_dimension = []  
 y\_dimension = []  
 for idx in indices:  
 x\_dimension.append(x[idx][0])  
 y\_dimension.append(x[idx][1])  
 plt.scatter(x\_dimension, y\_dimension)  
 plt.scatter(centers[i][0], centers[i][1], marker="^", s=200)  
plt.title("K-Means Clustering")  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.show()

K=2

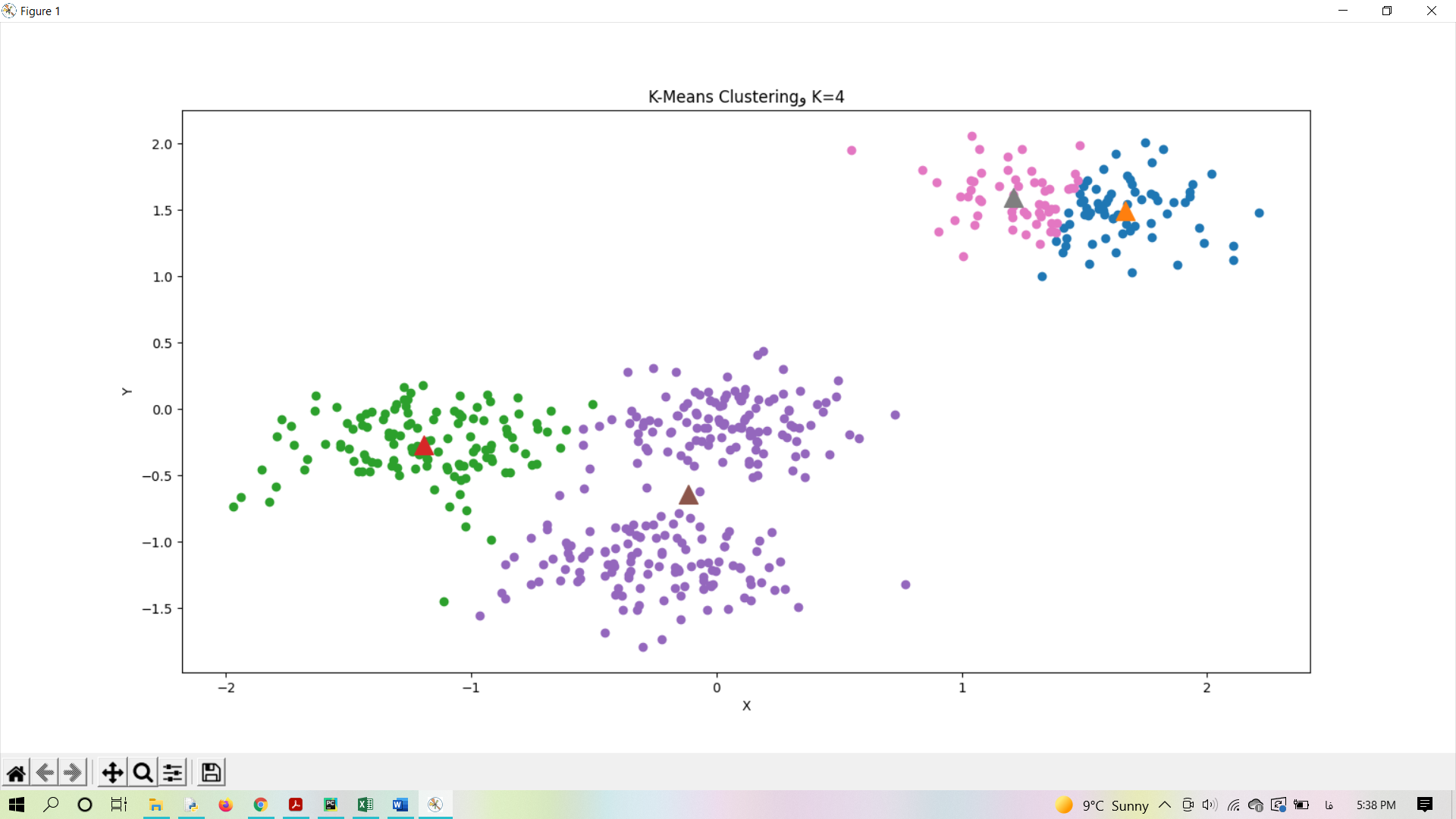


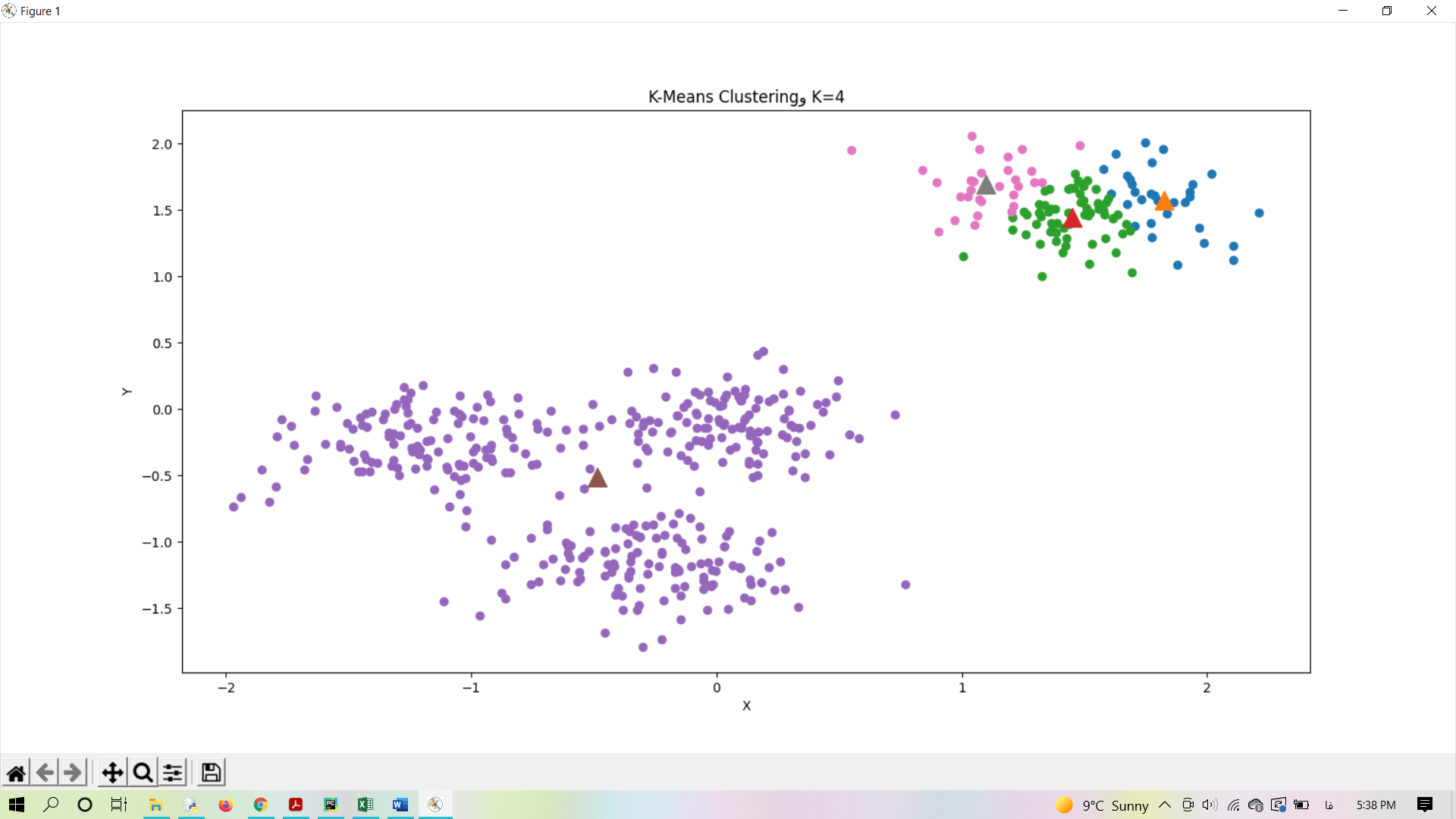
K=3



K=4







برای دو و سه خوشه، پس از چند بار اجرا نحوه ی شناسایی خوشه ها تقریبا مشابه یکدیگر بود. اما برای 4 خوشه، این الگوریتم، در اجراهای مختلف به روش های متفاوتی خوشه بندی کرد، که یک علت آن مراکز اولیه ای است که به صورت تصادفی انتخاب می شدند و در این تعداد خوشه تاثیر آن ها بهتر نمایان شده است.