**K-mean**

We are importing[Numpy](https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/) for statistical computations,[Matplotlib](https://www.geeksforgeeks.org/matplotlib-tutorial/) to plot the[graph,](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/) and make\_blobs from sklearn.datasets.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

**Create the custom dataset with make\_blobs and plot it**

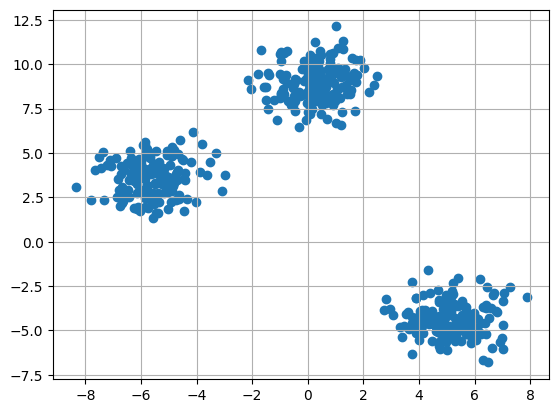
X,y = make\_blobs(n\_samples = 500,n\_features = 2,centers = 3,random\_state = 23)

fig = plt.figure(0)

plt.grid(True)

plt.scatter(X[:,0],X[:,1])

plt.show()



#### Initialize the random centroids

The code initializes three clusters for K-means clustering. It sets a random seed and generates random cluster centers within a specified range, and creates an empty[list](https://www.geeksforgeeks.org/list-cpp-stl/) of points for each cluster.

k = 3

clusters = {}

np.random.seed(23)

for idx in range(k):

center = 2\*(2\*np.random.random((X.shape[1],))-1)

points = []

cluster = {

'center' : center,

'points' : []

}

clusters[idx] = cluster

clusters

**Output:**

{0: {'center': array([0.06919154, 1.78785042]), 'points': []},  
 1: {'center': array([ 1.06183904, -0.87041662]), 'points': []},  
 2: {'center': array([-1.11581855, 0.74488834]), 'points': []}}

#### Plot the random initialize center with data points

plt.scatter(X[:,0],X[:,1])

plt.grid(True)

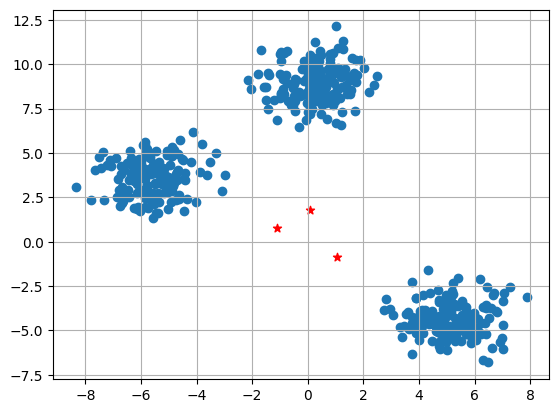
for i in clusters:

center = clusters[i]['center']

plt.scatter(center[0],center[1],marker = '\*',c = 'red')

plt.show()

**Output**:



The plot displays a scatter plot of data points (X[:,0], X[:,1]) with grid lines. It also marks the initial cluster centers (red stars) generated for K-means clustering.

#### Define Euclidean distance

def distance(p1,p2):

return np.sqrt(np.sum((p1-p2)\*\*2))

#### Create the function to Assign and Update the cluster center

The E-step assigns data points to the nearest cluster center, and the M-step updates cluster centers based on the mean of assigned points in K-means clustering.

#Implementing E step

def assign\_clusters(X, clusters):

for idx in range(X.shape[0]):

dist = []

curr\_x = X[idx]

for i in range(k):

dis = distance(curr\_x,clusters[i]['center'])

dist.append(dis)

curr\_cluster = np.argmin(dist)

clusters[curr\_cluster]['points'].append(curr\_x)

return clusters

#Implementing the M-Step

def update\_clusters(X, clusters):

for i in range(k):

points = np.array(clusters[i]['points'])

if points.shape[0] > 0:

new\_center = points.mean(axis =0)

clusters[i]['center'] = new\_center

clusters[i]['points'] = []

return clusters

#### Step 7: Create the function to Predict the cluster for the datapoints

def pred\_cluster(X, clusters):

pred = []

for i in range(X.shape[0]):

dist = []

for j in range(k):

dist.append(distance(X[i],clusters[j]['center']))

pred.append(np.argmin(dist))

return pred

#### Assign, Update, and predict the cluster center

clusters = assign\_clusters(X,clusters)

clusters = update\_clusters(X,clusters)

pred = pred\_cluster(X,clusters)

#### Plot the data points with their predicted cluster center

plt.scatter(X[:,0],X[:,1],c = pred)

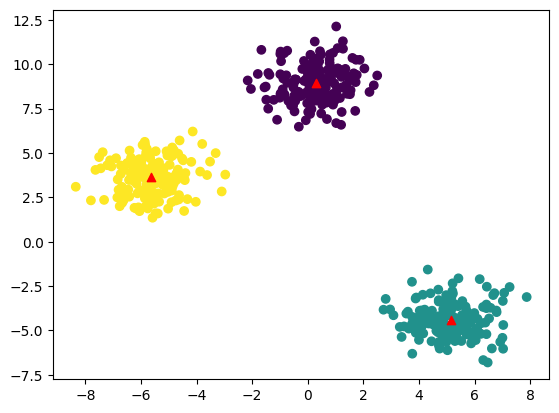
for i in clusters:

center = clusters[i]['center']

plt.scatter(center[0],center[1],marker = '^',c = 'red')

plt.show()

**Output**:



The plot shows data points colored by their predicted clusters. The red markers represent the updated cluster centers after the E-M steps in the K-means clustering algorithm.