Machine learning

K-MEANS

Morteza khorsand

- **Clustering**
  K-Means

How does the K-Means Algorithm Work?

Step-1: Select random K points or centroids as the center of clusters.

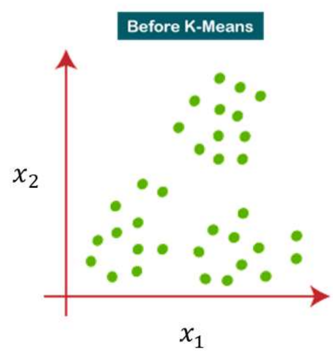
Step-2: Assign each data point to its closest centroid, which will form the predefined K clusters.

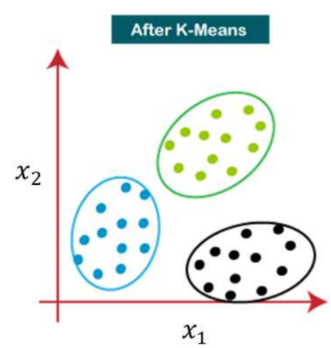
Step-3: Calculate the variance and place a new centroid of each cluster.

Step-4: Repeat the third step, which means reassigning each data point to the new closest centroid of each cluster.

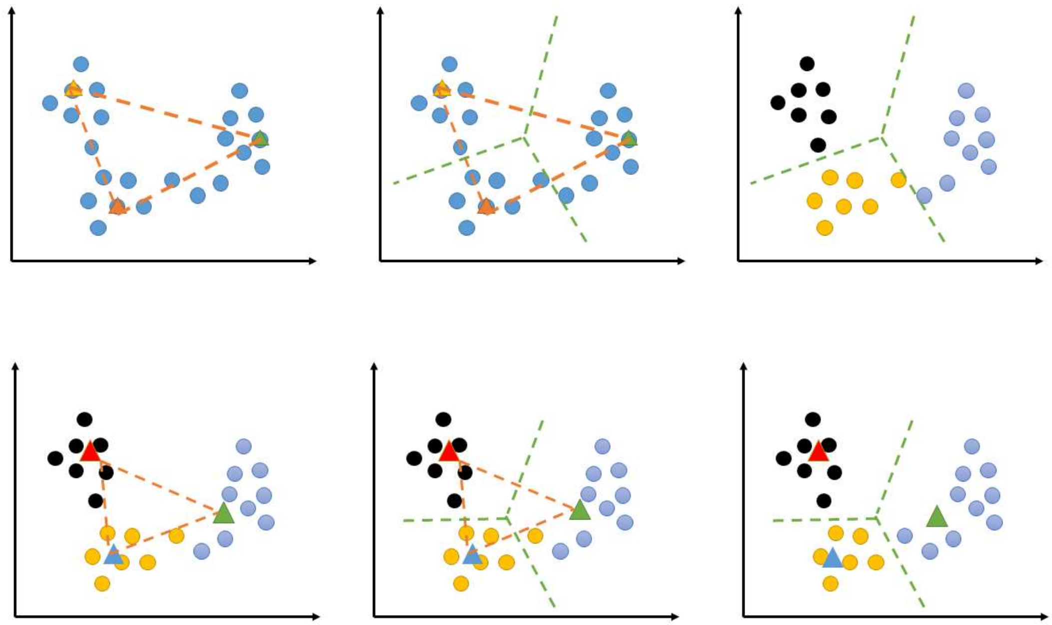
Step-4: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-6: The model is ready.





K - MEANS 2



K - MEANS 3

- **Pseudo code**

$randomly\ initiate\ K\ cluster\ centeroid \quad (\mu_1, \mu_2, \ldots \mu_k \in R^n)$

Repeat
{
  for i = 1 to m
  $c^{(i)} = argmin \left\| x^{(i)} - \mu_k \right\|$

  for k = 1 to k
  μ_k = average of points assigned to cluster K

}

Centroids = np.random.random ((k, n))

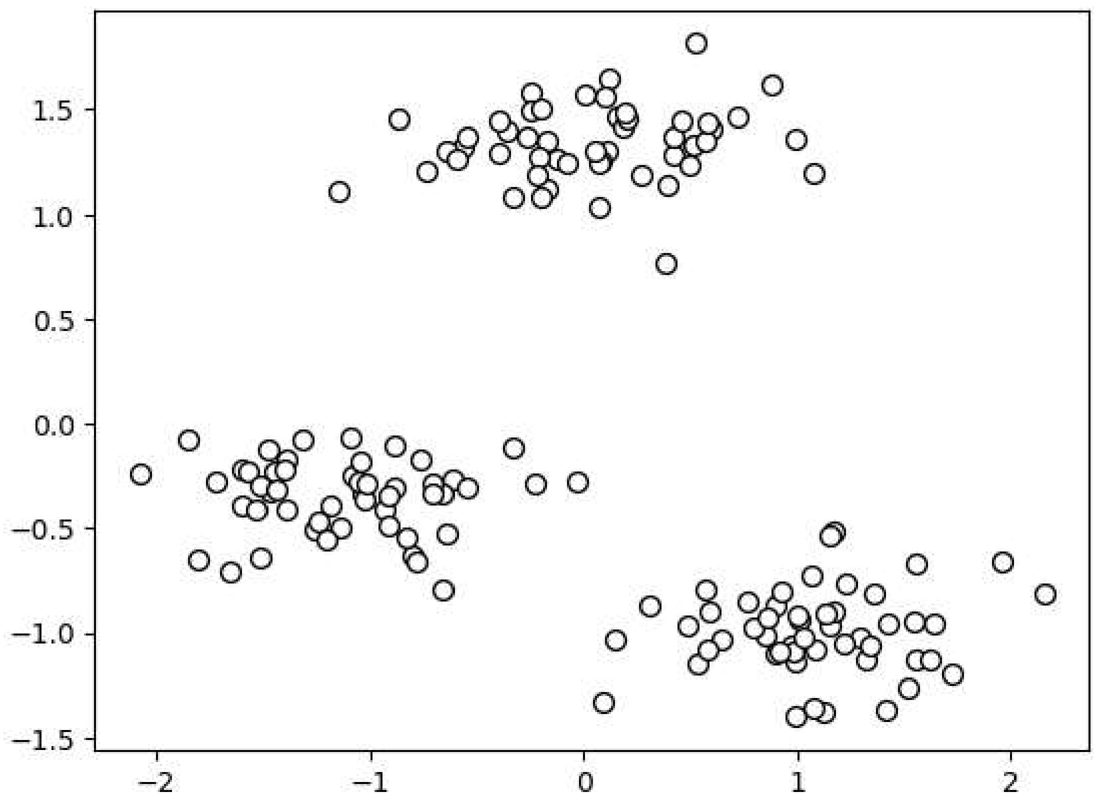
While True:
  for i in range(m):
    c[i] = np.argmin(np.linalg.norm(x[i] – centroids , axis = 1)
  for k in range(k):
    centroids[k] = np.mean(X[c==k] , axis = 0)
Termination condition

In [10]:
```
import os
os.environ["OMP_NUM_THREADS"] ='4'
```

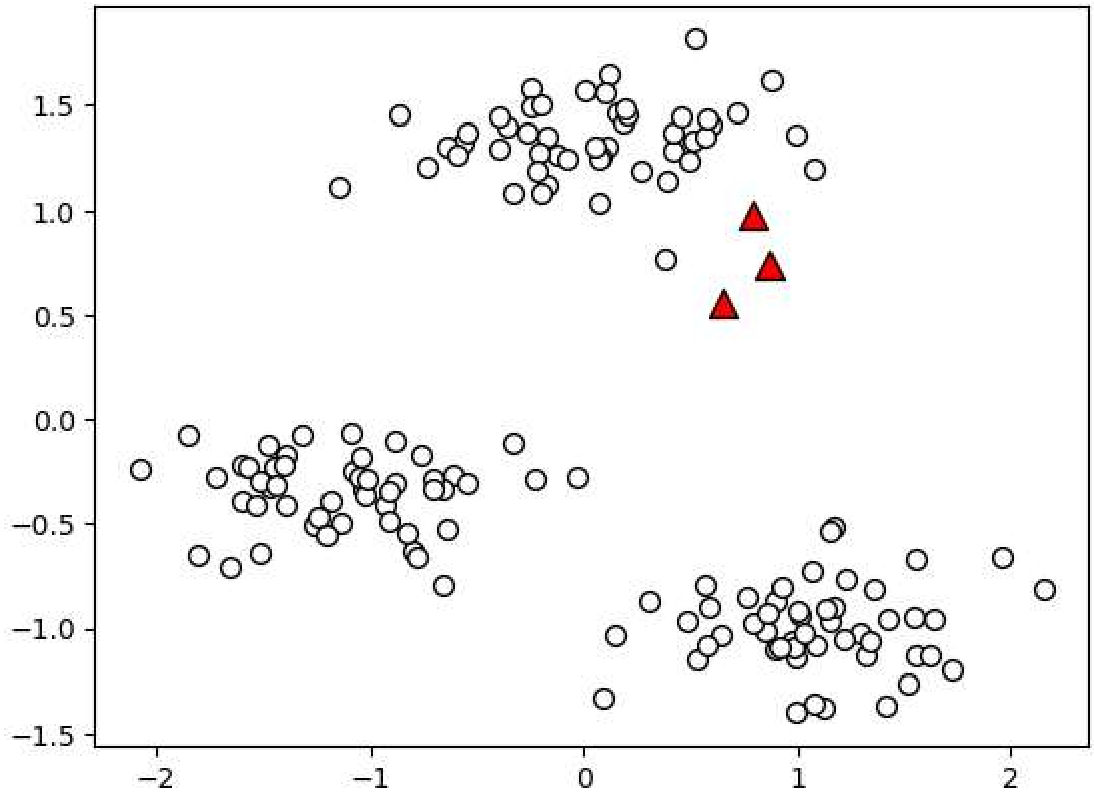
```
#C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarni
#warnings.warn
```

In [11]:
```
#import libraries and data
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
# create random data
X , y = make_blobs(n_samples=150, centers=3, cluster_std=1.2, random_state=10)
```

In [12]:
```
# Normalize X
mu = X.mean(axis=0)
std = X.std(axis=0)
X = (X - mu) / std

# plot data
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w')
plt.show()
```



In [4]:
```
m , n= X.shape
K=3
initial_centroids= np.random.rand(K,n)


#plot centroids
plt.scatter(initial_centroids[:, 0], initial_centroids[:, 1], edgecolors='k', s=100, c
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w'  )
plt.show()
```



In [13]:
```
centroids=initial_centroids.copy()
print(centroids)
```

```
[[0.87297331 0.73952191]
 [0.65928464 0.56123203]
 [0.79516162 0.97372791]]
```

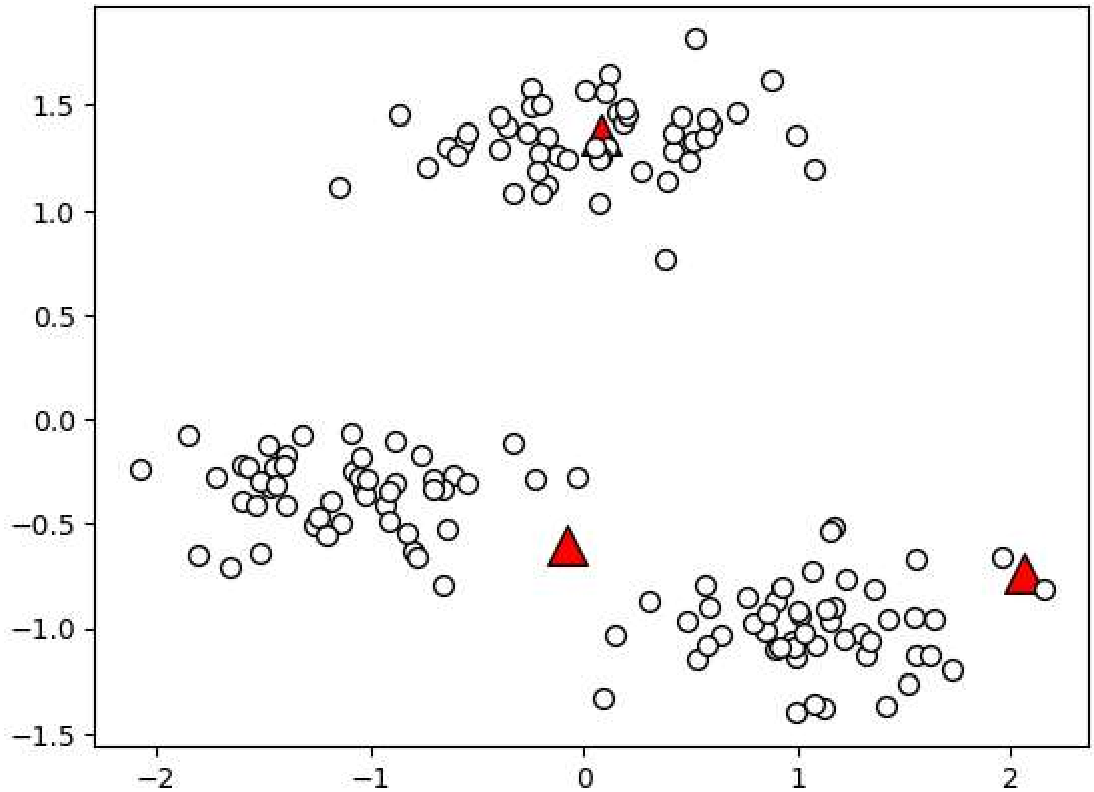
In [6]:
```
cluster_ids=np.array([np.argmin(np.linalg.norm(X[i] - centroids , axis = 1)) for i in


print(cluster_ids)
```

```
[1 2 2 1 1 2 1 2 1 1 2 2 1 1 2 2 2 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 1 1 1
 2 0 2 1 1 2 2 1 1 2 1 1 2 2 1 1 1 1 1 1 1 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 0 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 2 2 1 2 1 2 2
 1 2 2 1 1 1 2 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1 2 2 2 1 1
 1 1]
```

In [14]:
```
for k in range(K):
    centroids[k] = np.mean(X[cluster_ids==k], axis=0)    #ریس ایکس روبکشیم معلوم میشه
centroids
```

Out[14]:
```
array([[ 2.05851469, -0.73481985],
       [-0.0775947 , -0.60107719],
       [ 0.08255718,  1.36477202]])
```

In [15]:
```
plt.scatter(centroids[:, 0], centroids[:, 1], edgecolors='k', s=200, c='red' , marker
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w'  )
plt.show()
```



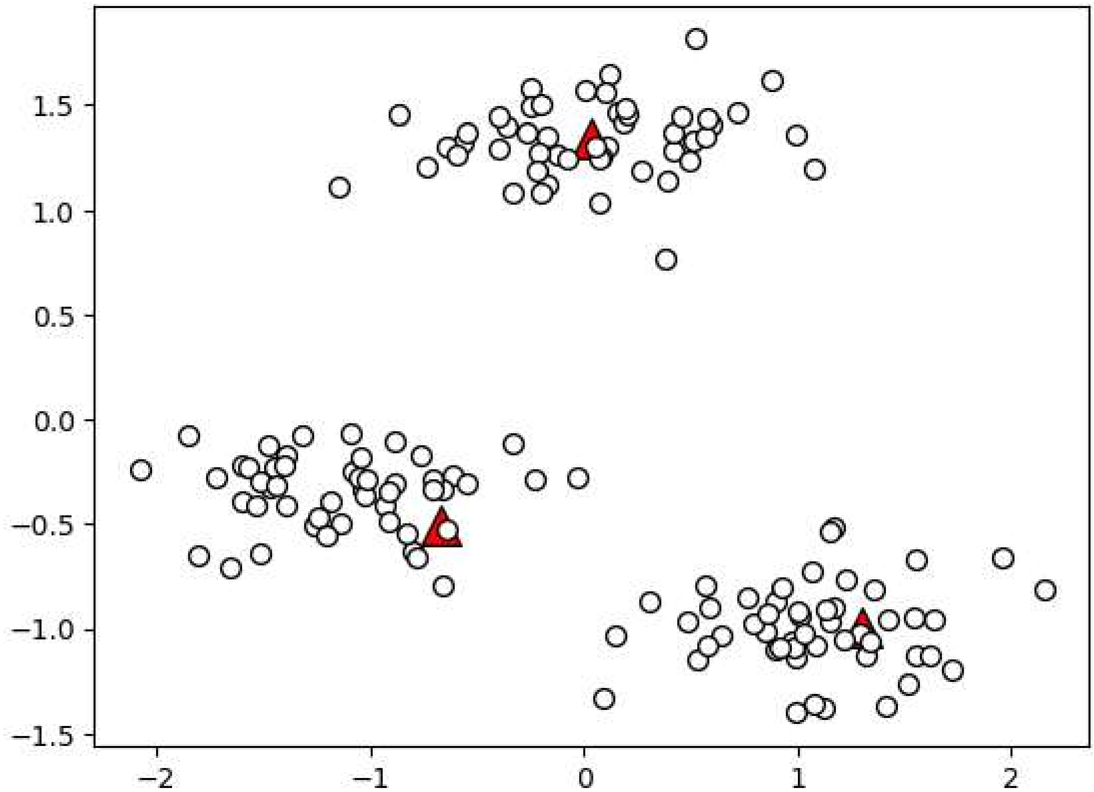
In [16]:
```
cluster_ids=np.array([np.argmin(np.linalg.norm(X[i] - centroids , axis = 1)) for i in


print(cluster_ids)
```

```
[1 2 2 1 1 2 1 2 0 0 2 2 1 1 2 2 2 0 2 1 1 0 1 1 1 1 0 1 2 2 0 1 1 2 0 1 0
 2 0 2 1 1 2 2 0 1 2 2 1 2 2 1 1 0 0 0 1 1 2 2 2 0 2 0 1 1 1 0 1 1 1 0 1 0
 1 0 1 0 1 2 0 1 0 1 2 1 2 1 2 1 1 1 0 2 2 2 1 2 1 1 1 0 1 1 2 2 1 2 1 2 2
 1 2 2 0 0 0 2 0 1 1 1 1 0 2 2 1 0 2 1 1 1 1 2 0 1 1 1 2 1 0 2 1 2 2 2 1 1
 0 1]
```

In [17]:
```
for k in range(K):
    centroids[k] = np.mean(X[cluster_ids==k], axis=0)    #ریس ایکس روبکشیم معلوم میشه
centroids
```

Out[17]:
```
array([[ 1.30323512, -0.99493691],
       [-0.67151137, -0.50937076],
       [ 0.03969005,  1.33921518]])
```

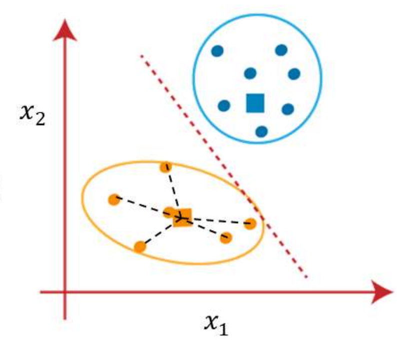
In [18]:
```
plt.scatter(centroids[:, 0], centroids[:, 1], edgecolors='k', s=200, c='red' , marker
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w'   )
plt.show()
```



μ_k : center of clusters

$c^{(i)}$: index of cluster assigned to $x^{(i)}$

$\mu_c^{(i)}$: center of cluster assigned to $x^{(i)}$



Goal function : J $(c^{(1)}, c^{(2)}, c^{(3)} \dots c^{(m)}, \mu_1, \mu_2, \mu_3 \dots \mu_k)$

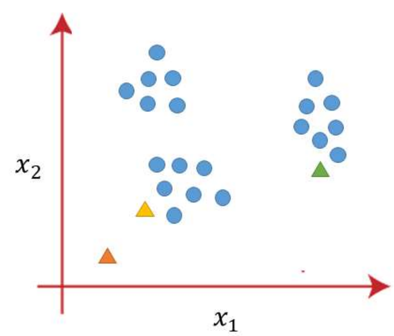
$$\text{WCSS} = \frac{1}{m}\sum_{i=1}^{m} \left\| x^{(i)} - \mu_k \right\|^2$$

WCSS= $\sum_{\text{Pi in Cluster1}}$ distance$(P_i\ C_1)^2$ +$\sum_{\text{Pi in Cluster2}}$distance$(P_i\ C_2)^2$+$\sum_{\text{Pi in CLuster3}}$ distance$(P_i\ C_3)^2$

■ Problem

Select from existing data

C= np.random.permutation(x) [: k]

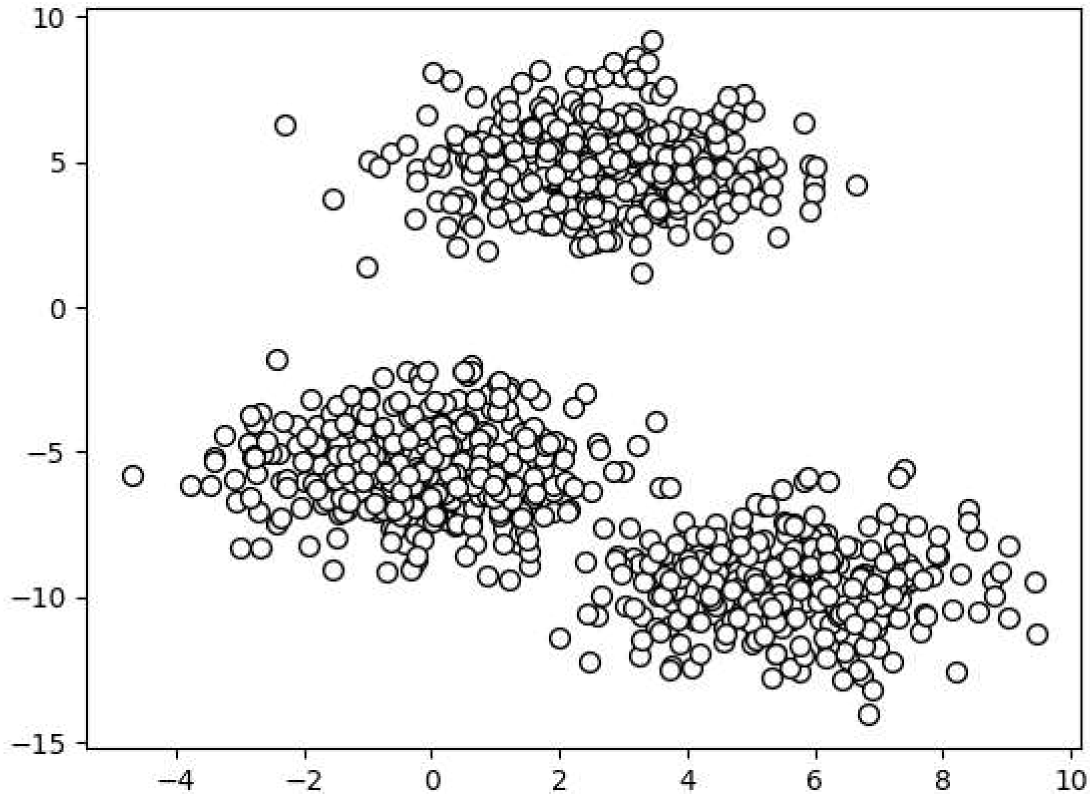


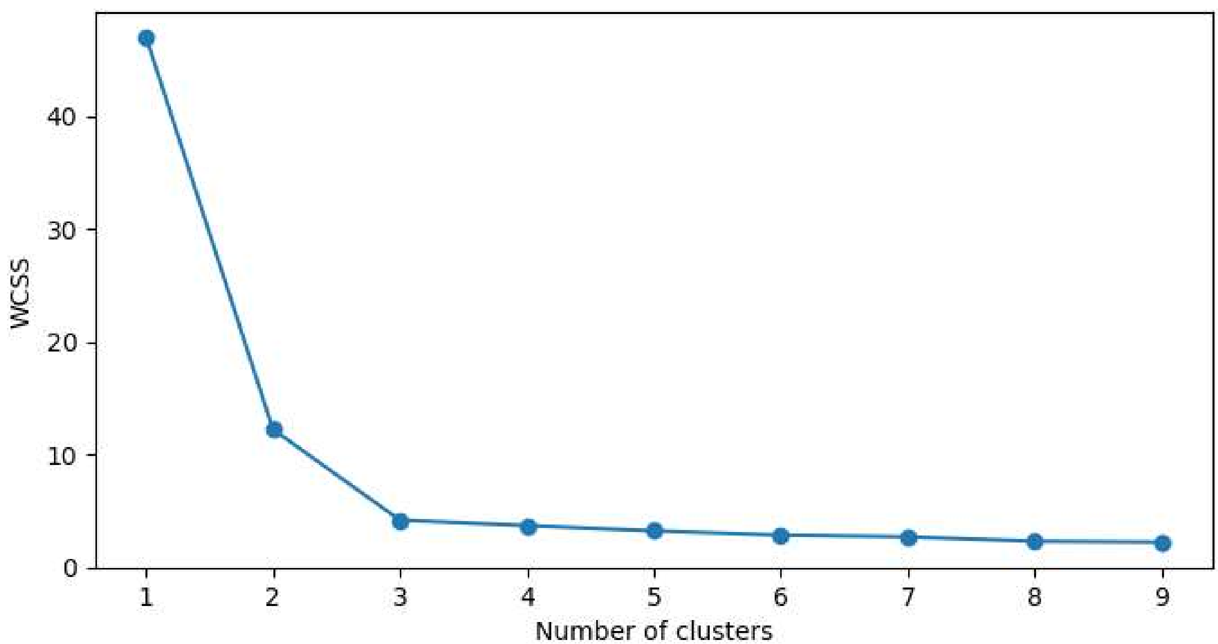
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
```

In [2]:
```python
def kmeans(X, initial_centroids):
    m = X.shape[0]
    K = initial_centroids.shape[0]
    centroids = initial_centroids
    initial_cluster_ids =np.zeros((m,))
    cost=0

    while True:
        cluster_ids = np.array([np.argmin(np.linalg.norm(X[i] - centroids, axis=1)) fo

        # update cluster centers
        for j in range(K):
            centroids[j] = np.mean(X[cluster_ids==j], axis=0)    #بگیر بذار به عنوان مرکز

         # stop
        if np.all(cluster_ids == initial_cluster_ids):
            for z in range(K):
                cost+=1/m * (np.linalg.norm(X[cluster_ids==z] - centroids[z])**2)
            return cost , initial_cluster_ids , centroids
        else:
            initial_cluster_ids = cluster_ids
```

In [3]:
```python
X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.5, random_state=10)
# plot data
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w');




costs=[]
best_centeroids=[]
final_ids=[]
for i in range(1 , 10):
    initial_centroids= np.random.permutation(X[ : i])
    A= kmeans(X , initial_centroids)
    costs.append(A[0])
    final_ids.append(A[1])
    best_centeroids.append(A[2])

print(costs)
print("--------------------------------------------------------")
print(best_centeroids[2])
print("--------------------------------------------------------")
print(final_ids[2])


plt.figure(figsize=(8, 4))
plt.plot(range(1, 10), costs, marker='o')
plt.xticks(range(1, 10))
plt.xlabel('Number of clusters')
plt.ylabel("WCSS")
plt.show()
```

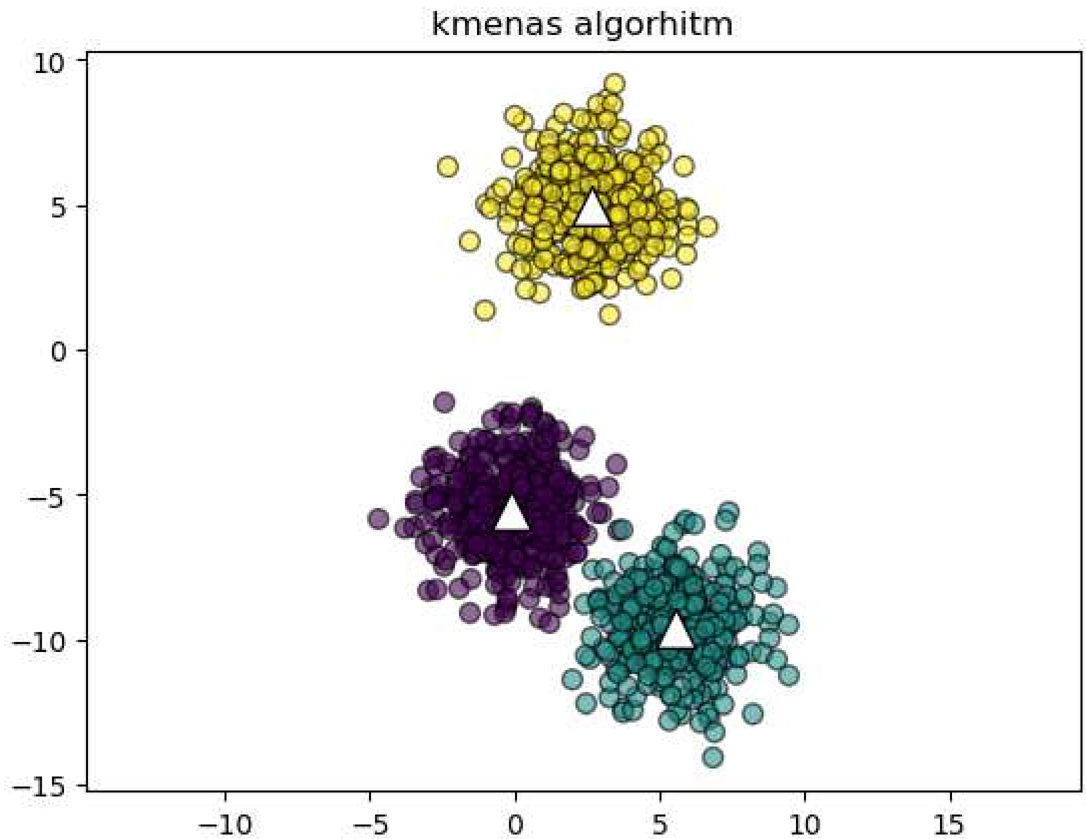
```
[47.029062199797174, 12.21527718761187, 4.167450222191264, 3.6894595293854398, 3.2242
86583361303, 2.843484126290323, 2.6948171538429073, 2.3008401012827138, 2.22501552576
16107]
-------------------------------------------------------
[[ 2.61164305  4.95788186]
 [ 5.51998791 -9.57304198]
 [-0.13936238 -5.54381213]]
-------------------------------------------------------
[1 0 2 2 0 2 2 2 2 2 1 0 1 1 2 2 2 1 1 1 2 1 2 2 0 0 1 0 1 1 1 1 1 1 1 0 0
 1 0 0 0 2 0 0 1 0 2 2 1 1 1 2 0 2 0 2 2 1 0 0 2 2 1 0 2 1 0 2 0 0 0 2 2 1
 2 2 1 1 1 2 2 2 0 2 0 1 0 2 2 0 0 1 1 0 2 1 0 1 1 0 0 2 0 0 2 1 2 0 1 2 2
 0 1 0 2 1 0 2 2 0 1 1 1 1 1 1 0 2 0 0 0 2 0 2 2 2 2 1 2 2 1 0 2 1 0 1 2 0
 0 2 0 1 1 2 2 0 2 2 0 2 0 2 1 1 0 0 0 0 1 2 0 2 2 2 1 1 2 1 1 1 2 2 1 2 0
 0 2 2 2 0 2 2 1 1 0 1 0 0 2 2 2 1 0 2 1 1 0 0 0 2 0 0 0 0 0 0 1 1 0 2 0 2
 1 1 2 0 2 0 1 2 0 1 0 1 2 2 1 0 2 2 0 2 0 1 1 0 2 2 2 0 2 0 1 0 0 0 0 2 0
 0 1 1 0 2 0 1 1 1 0 1 1 0 2 1 2 1 0 0 1 1 1 0 2 0 0 1 1 1 1 2 2 0 0 0 2 0
 2 1 1 2 2 1 2 1 1 1 2 2 1 0 1 1 2 1 1 0 2 1 2 2 2 0 0 1 2 1 0 2 1 2 0 2 1
 2 1 1 1 1 2 1 1 0 2 0 1 0 0 0 1 0 1 1 1 0 0 2 2 1 0 2 2 2 1 2 1 0 2 2 0 2
 2 2 1 0 2 2 1 1 2 1 1 0 2 0 0 0 1 2 1 1 2 2 0 2 2 0 1 2 2 2 0 1 2 2 2 1 0
 2 0 2 1 1 2 2 2 1 2 1 1 0 2 1 2 1 2 1 0 0 0 0 2 1 2 1 1 0 0 2 0 2 1 0 1 1
 0 0 1 1 2 0 0 0 2 1 1 0 0 1 0 1 0 2 1 1 2 0 1 2 2 0 2 1 2 2 1 1 1 2 1 1 0
 0 2 1 2 2 1 1 1 2 0 2 1 2 2 2 1 2 0 1 1 2 2 2 1 1 1 1 2 2 0 1 0 0 1 1 2 0
 2 2 0 2 0 2 2 0 1 0 2 2 1 2 0 0 0 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 2 2 0 0 1
 1 0 1 0 0 1 1 2 1 0 1 0 1 0 0 2 1 1 2 1 2 2 2 0 0 0 2 0 1 0 2 1 1 1 0 0 0
 0 0 0 1 2 2 2 2 2 0 0 1 0 0 1 1 2 0 2 0 2 0 2 0 1 0 1 2 2 1 1 2 0 2 2 1 0
 1 1 2 0 0 0 1 0 1 0 2 0 1 0 0 1 1 2 2 0 2 2 0 1 2 2 1 0 1 0 1 0 1 2 2 2 2
 0 2 2 2 2 1 1 0 0 0 0 1 2 2 0 1 2 2 0 1 1 0 2 0 1 1 1 1 1 1 1 0 1 2 0 2 1
 0 2 2 2 2 1 0 1 2 1 0 2 0 2 0 2 0 0 0 0 0 1 0 2 1 1 0 1 1 1 2 0 1 1 1 2 0
 1 0 2 2 2 1 0 1 0 1 0 0 1 1 0 1 0 2 0 0 0 1 2 0 1 2 1 1 2 0 2 0 1 0 0 0 2
 0 2 0 2 0 0 0 0 1 2 1 2 1 2 1 2 0 0 2 0 2 1 0 2 0 1 2 2 0 1 0 0 1 1 1 0 2
 0 1 2 0 2 0 1 0 1 0 1 0 1 0 0 2 2 1 0 1 0 1 2 0 2 1 2 1 2 2 1 2 2 1 1 1 0
 1 1 2 1 2 1 1 2 2 0 2 0 1 2 0 0 1 2 2 1 2 1 1 1 0 2 1 1 1 0 2 1 2 2 2 1 0
 0 1 1 1 0 0 2 2 0 0 2 2 1 1 0 0 2 0 1 1 2 1 0 0 0 1 2 0 0 1 1 0 1 1 1 2 2
 2 1 0 1 0 0 0 1 0 1 1 1 2 0 2 1 1 0 1 2 0 2 2 2 1 0 0 0 2 0 0 0 2 0 1 2 2
 0 1 1 2 0 1 1 0 2 1 1 2 2 0 1 2 0 1 0 1 2 0 0 0 2 0 2 2 1 1 2 0 1 2 1 0 0
 2]
```





In [22]:

```
centroids= best_centeroids[2]
ids=final_ids[2]

K = centroids.shape[0]

plt.figure()
plt.scatter(X[:, 0], X[:, 1], s=50,c=ids , edgecolors='k', alpha=0.6)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='^', s=200, c="white", edgecolors
plt.title("kmenas algorhitm")
plt.axis('equal')
plt.show()
```

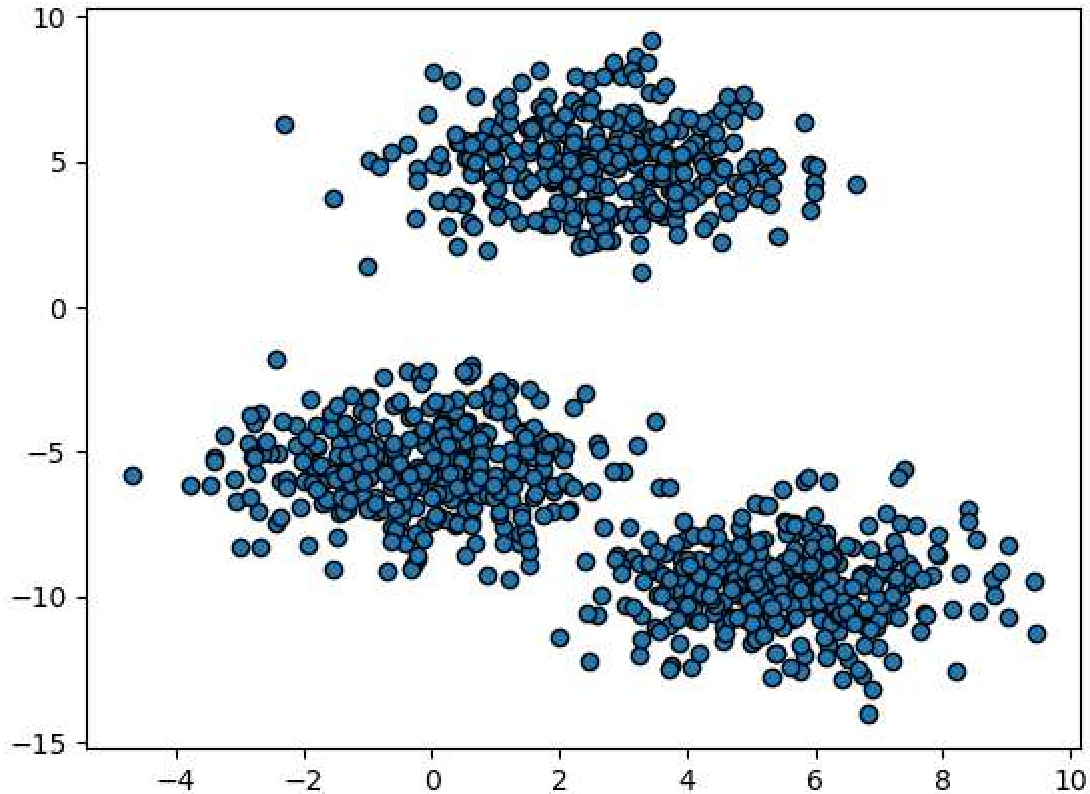


sklearn

In [23]:
```
from sklearn.cluster import KMeans
import pandas as pd
#the first two one(MinMaxScaler,StandardScaler) can scale and fit the data , they are
#the third one(scale) scale the data,  that is a method.
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import scale

import matplotlib.pyplot as plt
```

In [24]:
```
X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.5, random_state=10)



plt.scatter(X[:, 0], X[:, 1],  cmap=plt.cm.Set1, edgecolor="k")
```

Out[24]:
```
<matplotlib.collections.PathCollection at 0x1c40e2b7fa0>
```



In [26]:
```
X1_normalized= scale(X[:, 0] , axis= 0 , with_mean= True , with_std= True)
X2_normalized= scale(X[:, 1] , axis= 0 , with_mean= True , with_std= True)
```

In [27]:
```
d = {'first_value': X1_normalized, 'second_value':X2_normalized}
df=pd.DataFrame(d)
df
```

Out[27]:

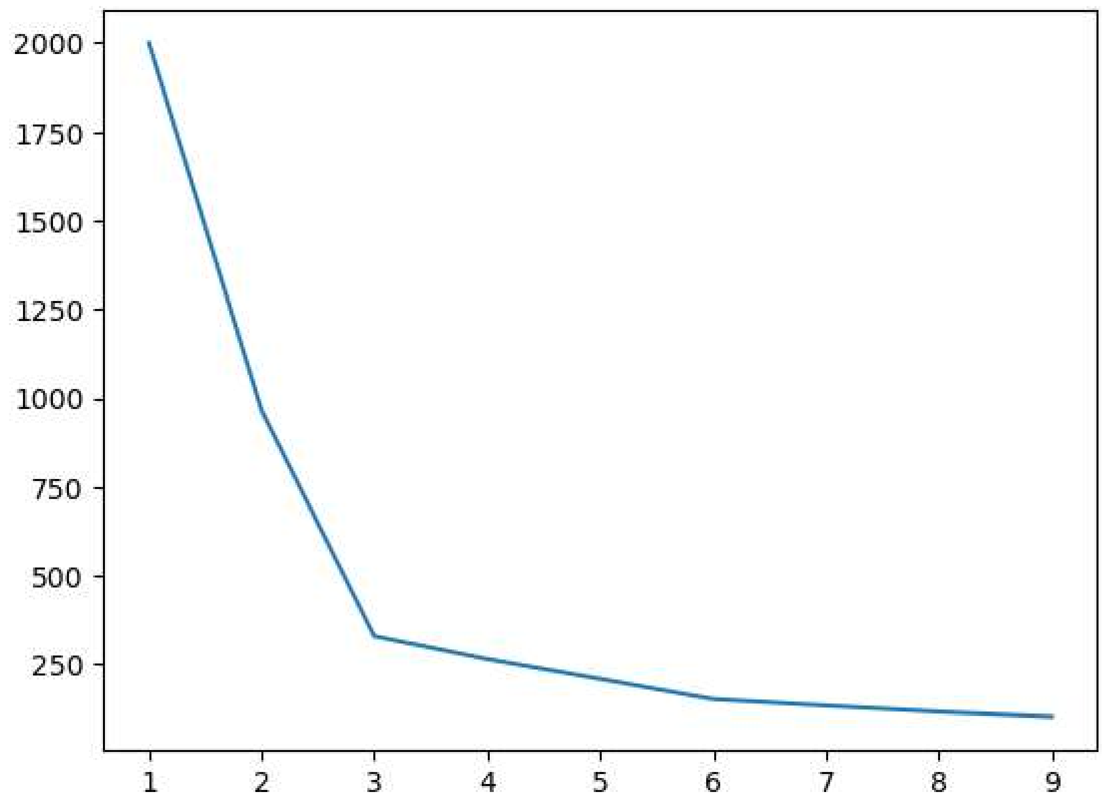
	first_value	second_value
0	0.541071	-1.070662
1	-0.173577	0.981308
2	-0.983675	-0.465678
3	-0.904900	-0.067413
4	-0.567329	1.491656
...	...	...
995	-1.446594	0.053757
996	1.100001	-0.660024
997	1.224302	1.304300
998	-0.868375	1.108034
999	-0.439211	-0.174590

1000 rows × 2 columns

In [28]:
```
wcss = []
for i in range(1,10):
    km= KMeans(n_clusters= i )
    km.fit(df[["first_value","second_value"]])
    wcss.append(km.inertia_)

wcss
```

Out[28]:
```
[1999.9999999999995,
 964.9482561631145,
 329.0090988040914,
 268.1966585182072,
 203.78400010045277,
 151.65696339152393,
 133.62066616440416,
 116.5614373429577,
 102.31137820719674]
```

In [18]:
```
plt.plot(range(1, 10),wcss)
plt.show()
```



In [29]:
```
km= KMeans(n_clusters= 3 )                                          #construc
y_predict=km.fit_predict (df[["first_value","second_value"]])

df["predict"] = y_predict
df
```

Out[29]:

	first_value	second_value	predict
0	0.541071	-1.070662	1
1	-0.173577	0.981308	0
2	-0.983675	-0.465678	2
3	-0.904900	-0.067413	2
4	-0.567329	1.491656	0
...	...	...	...
995	-1.446594	0.053757	2
996	1.100001	-0.660024	1
997	1.224302	1.304300	0
998	-0.868375	1.108034	0
999	-0.439211	-0.174590	2

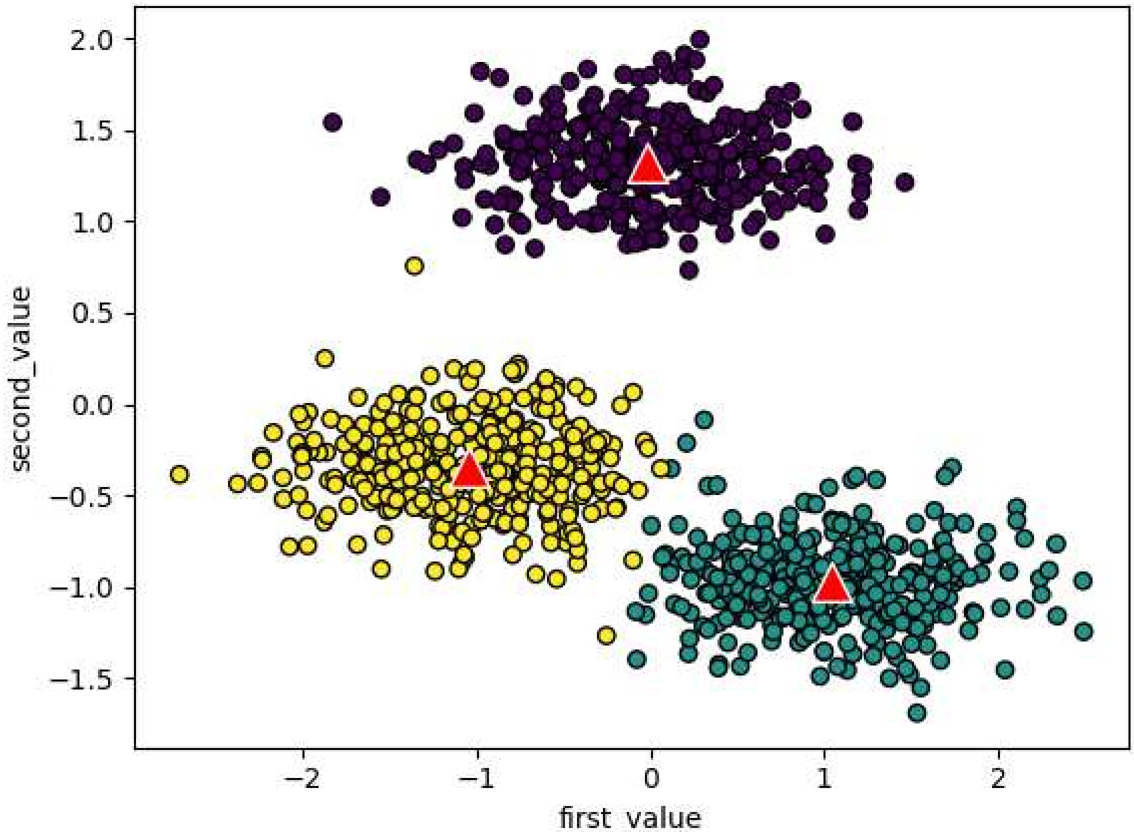
1000 rows × 3 columns

In [30]:
```
km.cluster_centers_
```

Out[30]:
```
array([[-0.0162935 ,  1.32842067],
       [ 1.04554222, -0.97308104],
       [-1.04184561, -0.34361576]])
```

In [31]:
```
plt.scatter(df["first_value"], df["second_value"],  c=df["predict"], edgecolor="k")
plt.scatter(km.cluster_centers_[: ,0] ,km.cluster_centers_[: ,1] , marker = "^" , s=2(
plt.xlabel("first_value")
plt.ylabel("second_value")
```

Out[31]:
```
Text(0, 0.5, 'second_value')
```

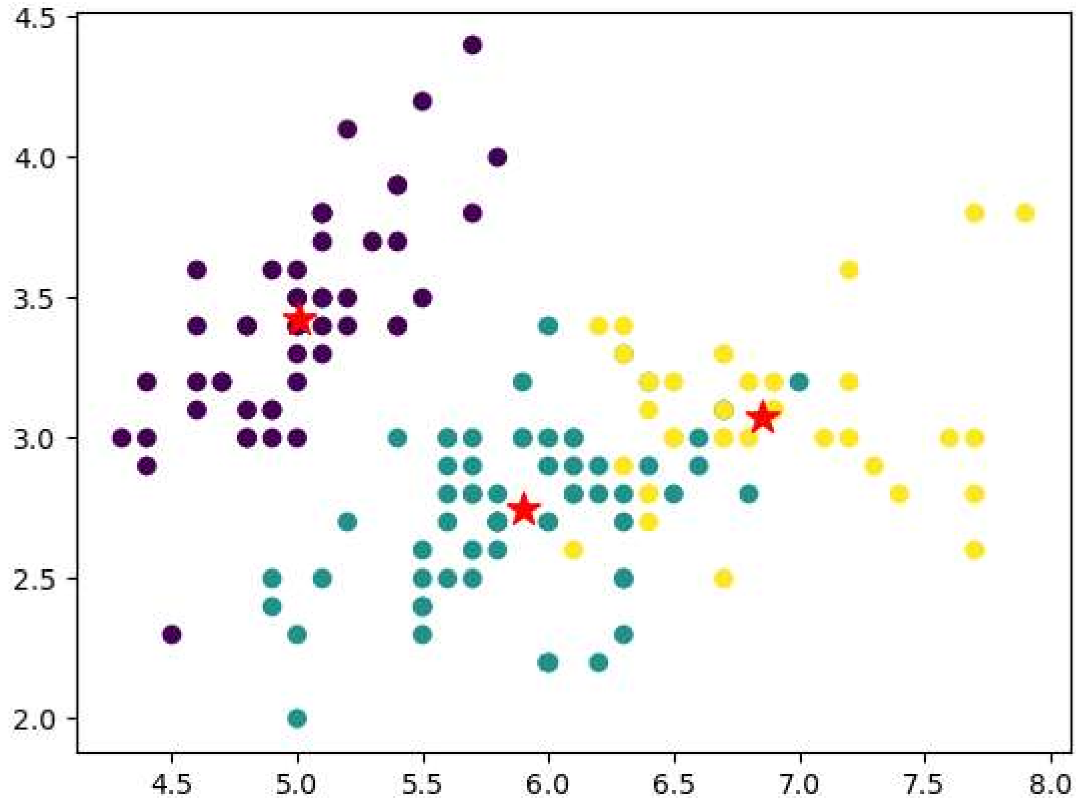


In [35]:
```
from sklearn.datasets import load_iris
```

In [36]:
```
iris=load_iris()
```

In [37]:
```
kmn=KMeans(n_clusters= 3)
kmn.fit(iris.data)
labels=kmn.predict(iris.data)
centroids=kmn.cluster_centers_
```

In [38]:
```
plt.scatter(iris.data[ : , 0] , iris.data[: ,1] , c= labels)
plt.scatter(centroids[ : , 0] , centroids[: ,1] , marker= "*" , c= "red" , s=150)
plt.show()
```




```
In [ ]:
```