```
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import math
import io
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files dataset.csv

- **dataset.csv**(text/csv) - 75 bytes, last modified: 4/30/2022 - 100%

```
Saving dataset.csv to dataset.csv
```

```
data = pd.read_csv(io.BytesIO(uploaded['dataset.csv']))
data
```

	X	Y
0	0.10	0.60
1	0.15	0.71
2	0.08	0.90
3	0.16	0.85
4	0.20	0.30
5	0.25	0.50
6	0.24	0.10
7	0.30	0.20

```
X = np.array(data)
```

```
c_x = np.array([0.1,0.3])
c_y = np.array([0.6,0.2])

centroids = np.array(list(zip(c_x,c_y)))
centroids
```

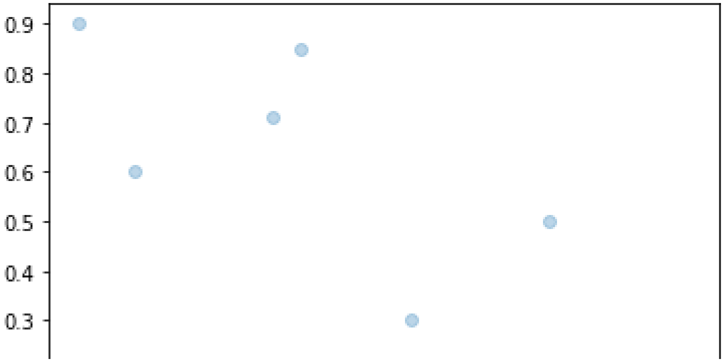
```
     array([[0.1, 0.6],
            [0.3, 0.2]])
```

```
class K_Means:
    def __init__(self, k=2, tol=0.001, max_iter=300):
        self.k = k
        self.tol = tol
        self.max_iter = max_iter

    def fit(self,data,centroids):

        self.centroids = {}

        for i in range(self.k):
            self.centroids[i] = centroids[i]

        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.k):
                self.classifications[i] = []

            for featureset in data:
                distances = [np.linalg.norm(featureset-self.
                classification = distances.index(min(distanc
                self.classifications[classification].append(

            prev_centroids = dict(self.centroids)
            for classification in self.classifications:
                self.centroids[classification] = np.average(

            optimized = True

            for c in self.centroids:
                original_centroid = prev_centroids[c]
                current_centroid = self.centroids[c]
                if np.sum((current_centroid-original_centroi
                    print(np.sum((current_centroid-original_
                    optimized = False

            if optimized:
                break

    def predict(self,data):
        distances = [np.linalg.norm(data-self.centroids[cent
        classification = distances.index(min(distances))
        return classification
```

```
model = K_Means()
model.fit(X, centroids)
```

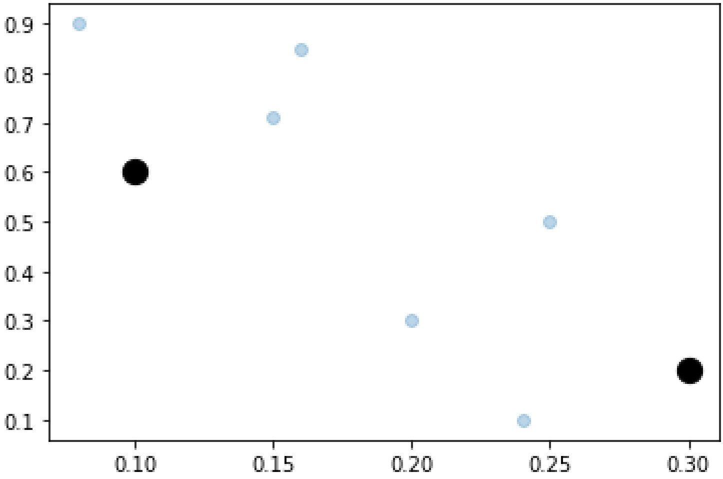
```
     66.66666666666666
```

```
## Data Points
plt.figure()
plt.scatter(X[:,0],X[:,1],alpha=0.3)
plt.show()
```

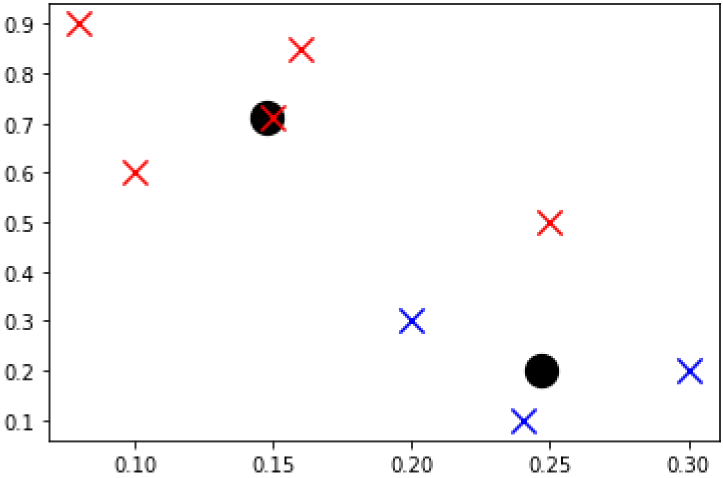


```
plt.figure()
plt.scatter(X[:,0],X[:,1],alpha=0.3)
plt.scatter(c_x,c_y, marker='o', c='black', s=150)
```

```
    <matplotlib.collections.PathCollection at 0x7fee0eeb8f
```



```
colors = ['r','b']

for centroid in model.centroids:
    plt.scatter(model.centroids[centroid][0], model.centroid
                marker="o", color="k", s=150, linewidths=5)

for classification in model.classifications:
    color = colors[classification]
    for featureset in model.classifications[classification]:
        plt.scatter(featureset[0], featureset[1], marker="x"

plt.show()
```



```
print("Point P6 belongs to cluster", model.predict([0.25,0.5
```

```
Point P6 belongs to cluster 0
```

```
print("Population of cluster2 is", len(model.classifications
```

```
Population of cluster2 is 3
```

```
print("Initial values of cluster centroids m1 and m2")
print("m1=",centroids[0])
print("m2=",centroids[1])

print("\nUpdated value of cluster centroids m1 and m2")
print("m1=",model.centroids[0])
print("m2=",model.centroids[1])
```

```
    print("m2=",model.centroids[1])
```

```
Initial values of cluster centroids m1 and m2
m1= [0.1 0.6]
m2= [0.3 0.2]

Updated value of cluster centroids m1 and m2
m1= [0.148 0.712]
m2= [0.24666667 0.2       ]
```

✓ 0s completed at 3:11 PM ● ✕