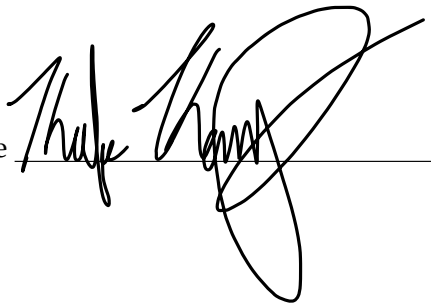
WORK ORIGINATION CERTIFICATION

By submitting this document, I, Kaitlyn Kamp, the author of this deliverable, certify that

1. I have reviewed and understood Regulation UCF 5.015 of the current version of UCF's Golden Rule Student Handbook available at http://goldenrule.sdes.ucf.edu/docs/goldenrule.pdf, which discusses academic dishonesty (plagiarism, cheating, miscellaneous misconduct, etc.)
2. The content of this Major Project report reflects my personal work and, in cases it is not, the source(s) of the relevant material has/have been appropriately acknowledged after it has been first approved by the course's instructional staff.
3. In preparing and compiling all this report material, I have not collaborated with anyone and I have not received any type of help from anyone but the course's instructional staff.

Signature  Date 8 Nov 22

1 Task 1

1.a

$$E = \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \| x_n - \hat{\mu}_k \|_2^2$$

$$\underline{x_n} \to H\underline{x_n} + c \qquad \underline{\mu_k} \to H\underline{\mu_k} + c$$

$$E = \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \| Hx_n + c - H\mu_k - c \|_2^2$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \| Hx_n - H\mu_k \|_2^2$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} (Hx_n - H\mu_k)^T (Hx_n - H\mu_k)$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} (Hx_n)^T Hx_n - 2(Hx_n)^T H\mu_k + (H\mu_k)^T H\mu_k$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \, x_n^T H^T H x_n - 2x_n^T H^T H \mu_k + \mu_k^T H^T H \mu_k$$

$H^T = H^{-1}$ so $H^T H = H^{-1} H = \underline{1}$

$$= \frac{1}{N} \sum_{k} \sum_{n} r_{n,k} \, x_n^T x_n - 2x_n^T \mu_k + \mu_k^T \mu_k$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \| x_n - \mu_n \|_2^2$$

$$E = E$$ ✓

1.b

$$x' = Ax$$

$$E = \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \| x_n - \mu_k \|_2^2$$

$$E = \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \| Ax_n - A\mu_k \|_2^2$$

$$E = \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} (Ax_n - A\mu_k)^T (Ax_n - A\mu_k)$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} (A(x_n - \mu_k))^T A(x_n - \mu_k)$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} (x_n - \mu_k)^T A^T A (x_n - \mu_k)$$

$$A^T A = S^{-1}$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} (x_n - \mu_k)^T S^{-1} (x_n - \mu_k)$$

$$= \frac{1}{N} \sum_{k=1}^{C} \sum_{n=1}^{N} r_{n,k} \| x_n - \mu_k \|_{S^{-1}}^2$$

2 Task 2

For this section the effects of different initialization methods are looked at. For each method K-Means was initialization 1000 times and the first loss was taken. All these losses are presented in histograms.

2.a

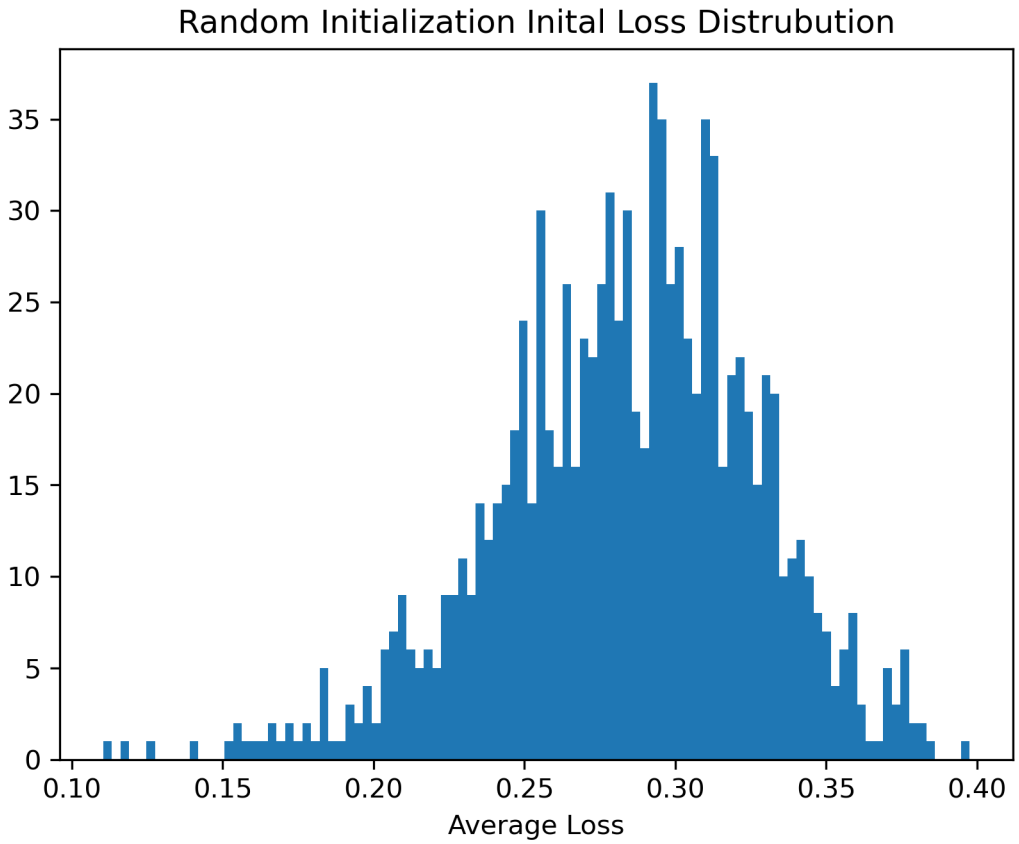


Figure 1: Distribution of losses for a random assignment. You can see that is is a quasi-normal distribution centered around 0.3.

2.b

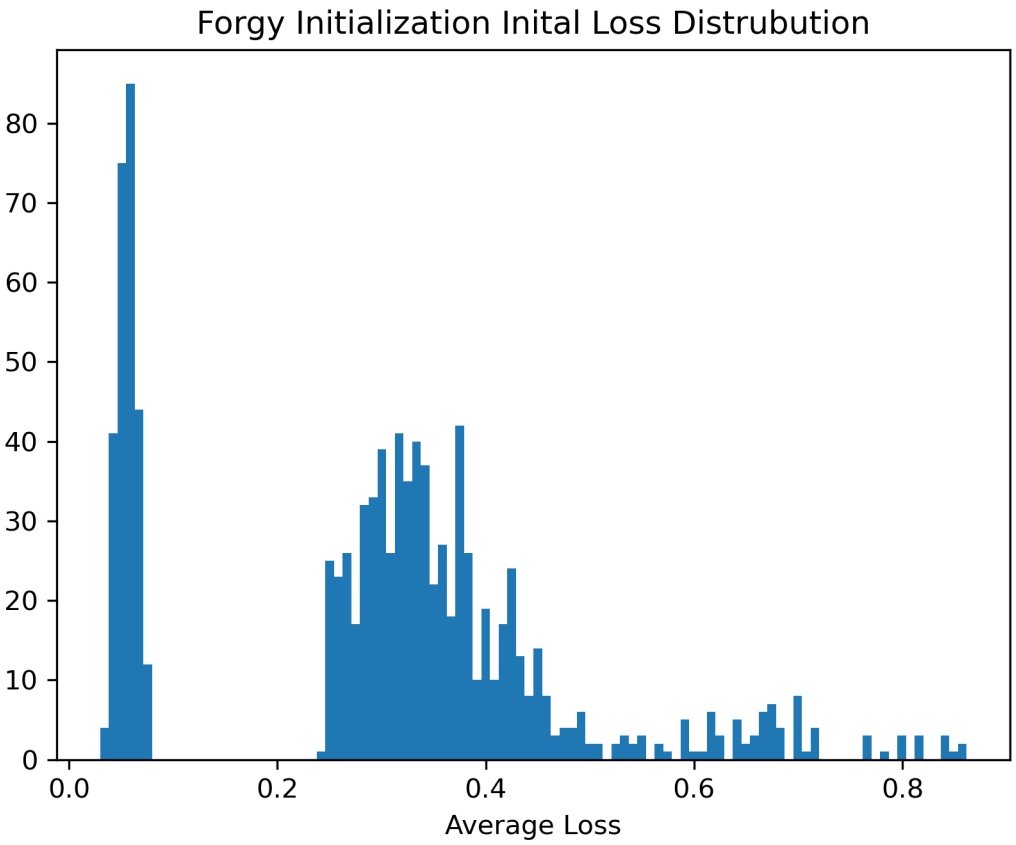


Figure 2: Distribution of losses for a forgy initialization. This displays a large concentration at low losses but a quasi-poisson distribution in the higher losses that trails off higher than the random initialization.

2.c

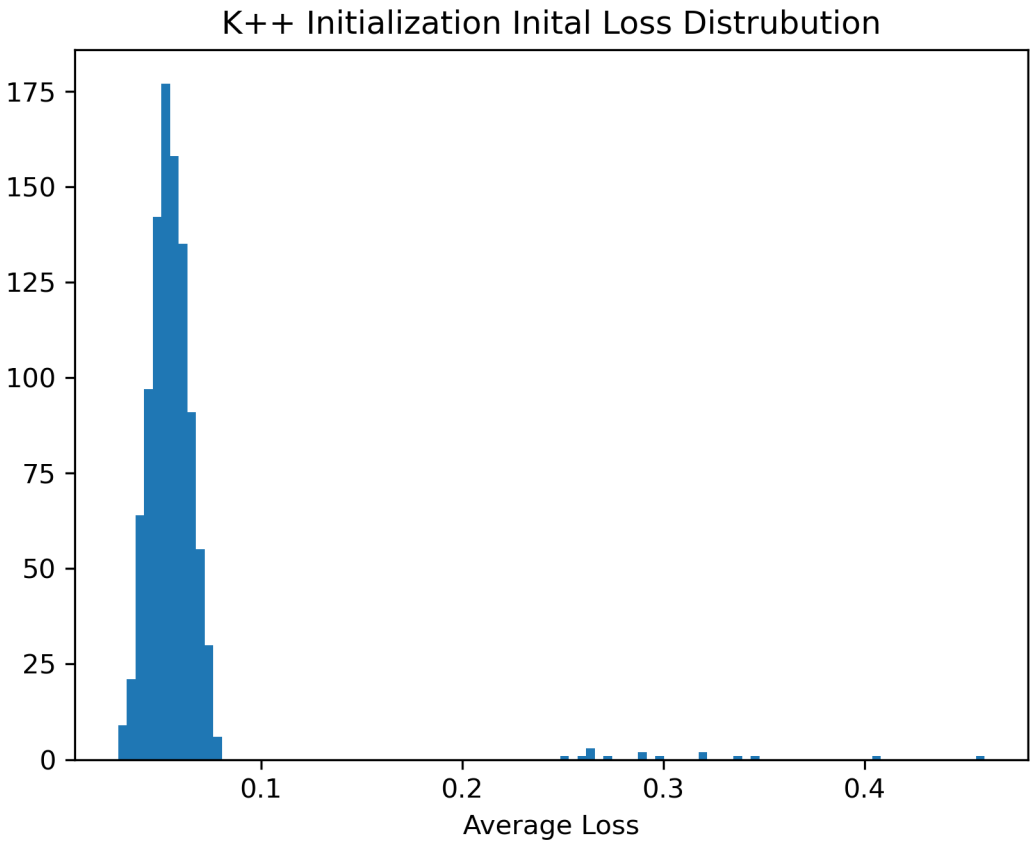


Figure 3: Distribution of K-meanss ++. There are some outliers to the right of the plot but the loss is concentrated in the lower values of the loss.

2.d

For ease of comparison all of the histograms are plotted together. We can see that K-means++ has the best initial loss with the peak being high and low with the occasional outlier. Forgy also has a high low peak however its residual goes pretty high whereas a random initialization does not go as high for initial loss. This is probably due to the forgy method being susceptible to choosing three means close to each other which is a hard "hole" to get out of where as the random initialization has a fall back of there are other samples to bring the means away from each other.

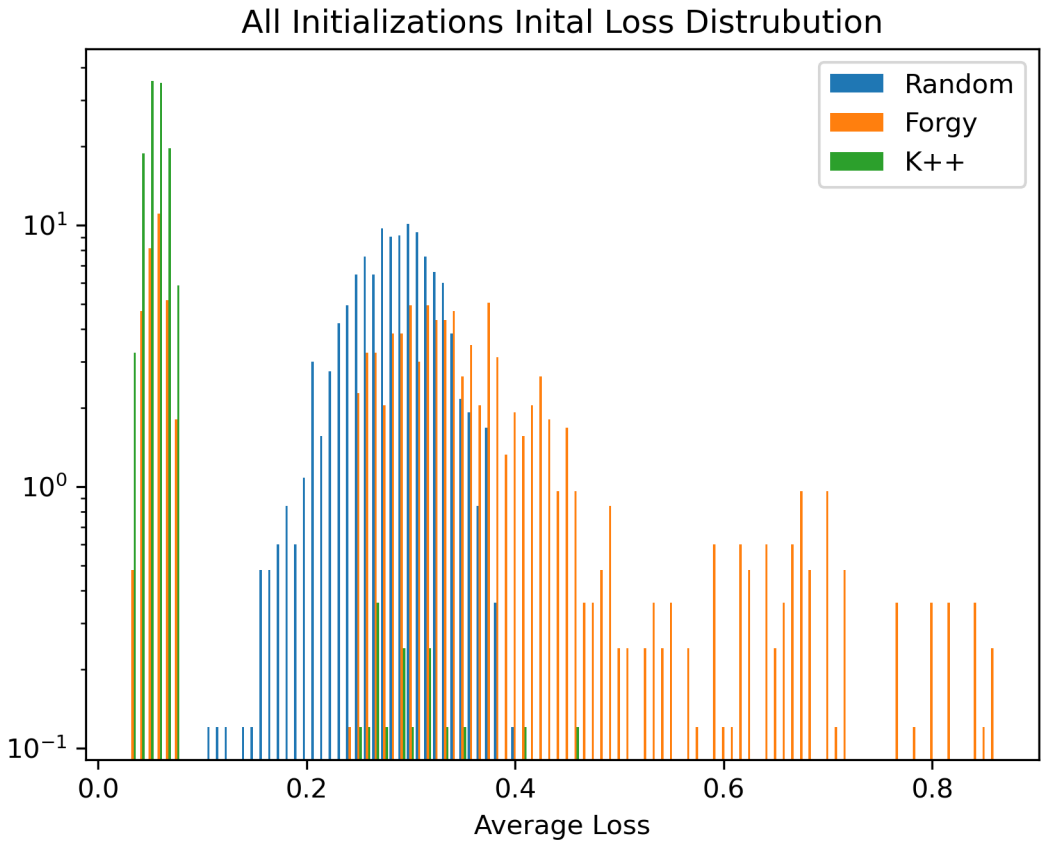


Figure 4: Plot of all the initial loss distributions to be able to compare to each other.

3 Task 3

In this section running K-means is looked at as well as the optimal number of means.

3.a

Initializing the K-means algorithm with K-means++ and running it with the Three data set with C=3 resulted in a quick convergence in two iterations. The first Log_{10}(Loss) was -0.3086 the converged with a Log_{10}(Loss) of -0.4450. The final locations of the averages can be seen by the stars on the plot.

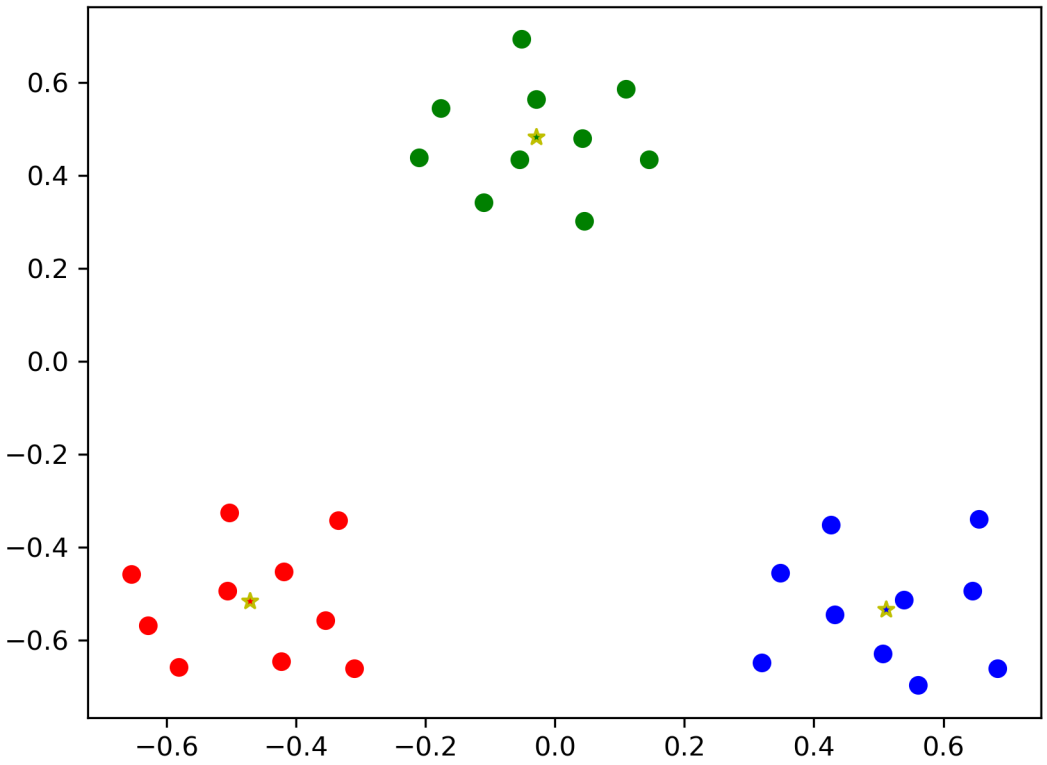


Figure 5: The conference of a K-means algorithm on the Three data set with K-means++ and C=3. The stars represent the class averages.

3.b

This subsection of the task looked at optimal numbers of clusters. I did this by looking at the average silhouette value for each of the clusters. The number of means with the highest average silhouette is the one that is the optimal number of averages. It can be seen in fig. 6 that C=3 is optimal with the average silhouette quickly decreasing as the number gets higher as well as it being lower at C=2.

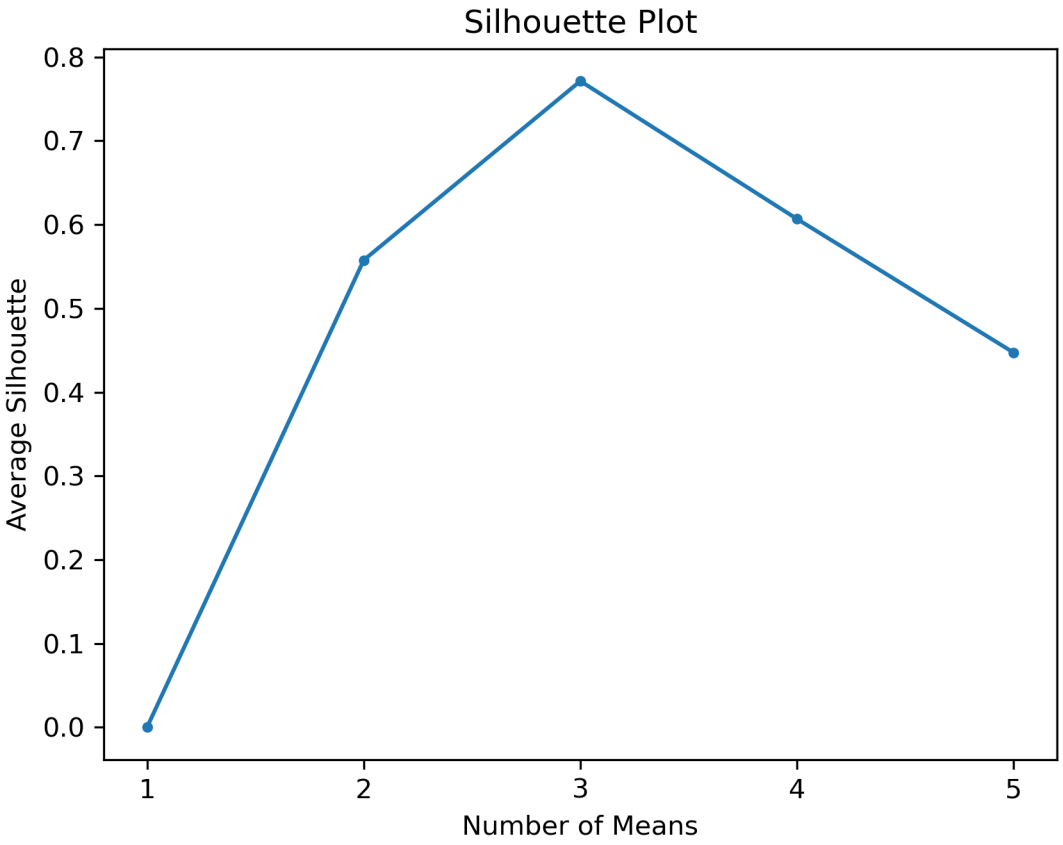


Figure 6: Plot of average silhouette vs number of means

3.c

This sub task looked at a distribution that wouldn't behave nicely with K-means. This data set has two groups but both of these groups have the same average one of them is just widely distributed. I ran K Means with K-means++ initialization on this set 10 times using 2 averages and then 5 averages. In this report I will show one of each plot but in the Notebooks at the end of the write up all plots are displayed. The first presented here in fig.7 is the run using C=2. With this these runs the groupings were not at all the two averages are seen to almost orbit in the space between the two groupings and never really getting the same grouping as another iteration. There was more constancy when the means ere upped C=5. Here each grouping can be seen in fig. 8 There was always a consistent group in the centre but the other 4 means would kinda rotate around the outside making each side to a square as seen in fig. 8. Occasionally some of the points in the middle would be "sucked" into a mean from the edge but this was an uncommon case. This is still not ideal since the other ones should be one class not 4 separate ones. It is more constant than C=2 though.

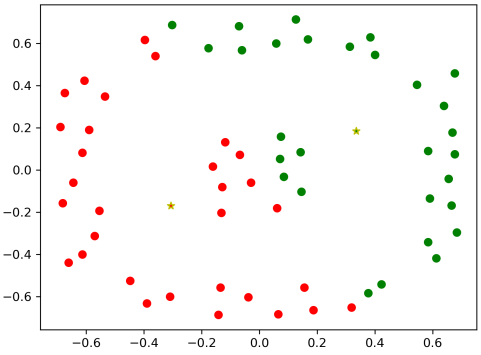


Figure 7: One iteration of K-means++ with C=2

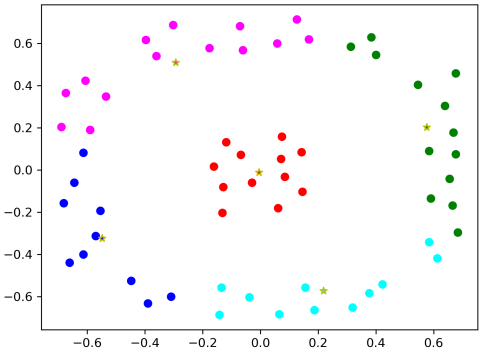


Figure 8: One iteration of K-means++ with C=5

4 Task 4

For this section of the mini project I investigated Vector Quantization for Lossy Compression using a K-Means algorithm I developed in the earlier in the mini project.

4.a

Vector quantization can easily be used for Lossy compression. The first step is to break the image into chunks, for this task I used 5x5 chunks. Next we taking all those chunks and vectorize them meaning to make them one long row. These features vectors can then be grouped by the K-means algorithm. Each of these vectors can then be replaced with the average value for its group. The image can then be rebuilt by reshaping the vectors into chunks and then piecing those chunks back together. This saves data since there are only C/N of the data C being the number of averages used in the K-mean algorithm and N the Number of chunks. This ratio also makes for a good measurement for compression as it is the ratio of approximated vectors/ components used to the original number of components.

4.b

I ran the VQ on the original image 4 times using C=1,10,100,1000. For each number of averages I ran the VQ 10 times and then rebuilt the image from the run with the lowest average loss.

Figure 9: This is the original Image put into the algorithm. In task4

C	Comp. Quality	Loss	PSNR
1	0.0001308	54.665	13.855
10	0.001308	22.722	17.667
100	0.01308	15.752	19.259
1000	0.1308	9.073	21.654

Table 1: Each of the runs the Loss, PSNE, and the Compression ratio.

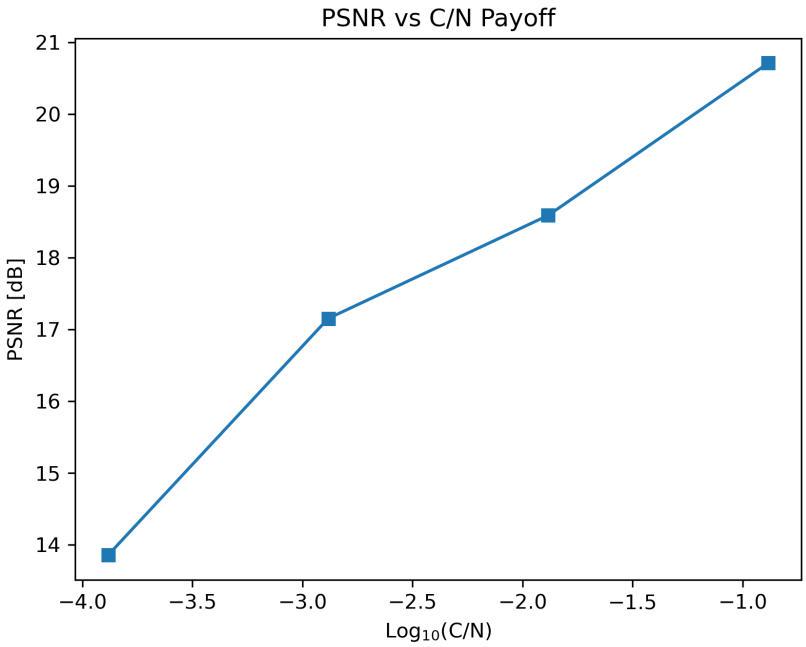


Figure 10: In the pay off between our proxy for compression $\log_{10}$ and the PSNR. As the compression gets smaller the PSNR gets smaller as well.

As expected as there are more means the compression quality and PSNR increases while the loss decreases. This is expected as since there are more means it is more likely for any point to be close to one. The increasing compression quality can be seen in the compressed pictures below where the lower compression quality gets the worse off the picture becomes. With only 1 mean C=1 we get the "average" 5 pixel chunk. This makes the image unrecognizable. However when I used 10 averages the image does become recognisable however very pixelated but the original picture is still recognizable. This trend continues with using 100 averages, The picture at first glance does look almost identical to the original image but it does look slightly pixelated if you zoom into it. With 1000 averages the process took about an hour for each run but the refiling picture looks nearly indistinguishable from the original.

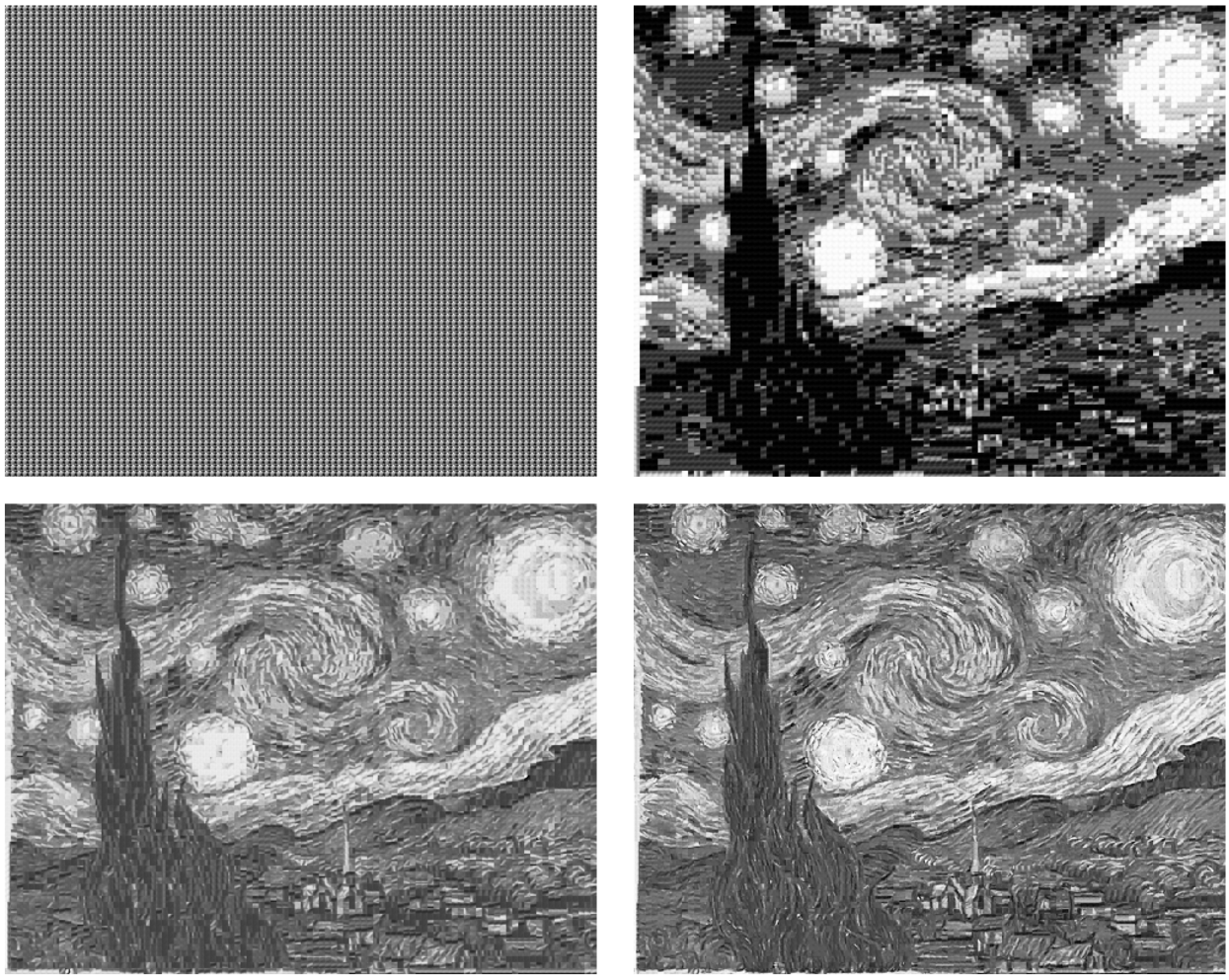


Figure 11: The compressed images after running VQ. The top row is C=1 and C=10 the bottom row is C=100 and C=1000

Attached are PDFs of python notebooks used

Task2

December 9, 2022

[3]:
```
import matplotlib.pyplot as plt
from astropy.io import ascii
import random
import numpy as np
import math
```

[4]:
```
class Kmean:
    def __init__(self,data,init='rand',C=3):
        #saving initlaizion params
        self.data=data
        self.init=init
        self.C=C
        self.data['y']=-1 #giving my data frame a column of labels for all -1
        self.colors=['r','g','b','cyan','magenta','purple','k'] #list of colors␣
 ↪for consoistancy
        self.Cov=np.cov(list(self.data.values())) #calculating covanacince
        if self.init=='rand': # Case for random
            self.Random()
            self.mean()
        elif self.init=='forgy': #case for Forgy
            self.Forgy()
            self.cluster() #cluster to asign labels
        elif self.init=='K++': #case for K mean ++
            self.Kpp()
            self.cluster() #cluster to asign labels
        else: #fall back case
            print('Initialization type not recognized defaulting to K++')
            self.init='K++'
            self.Random()
            self.mean()
            self.cluster() #cluster to asign labels
    def Random(self): #Random init
            self.mu=[[-1,-1]]*self.C #allocating an empy array
            y=np.random.randint(0,self.C,len(self.data['y'])) #assigning␣
 ↪ranomly labels
            test=1
            for i in range(self.C):
```

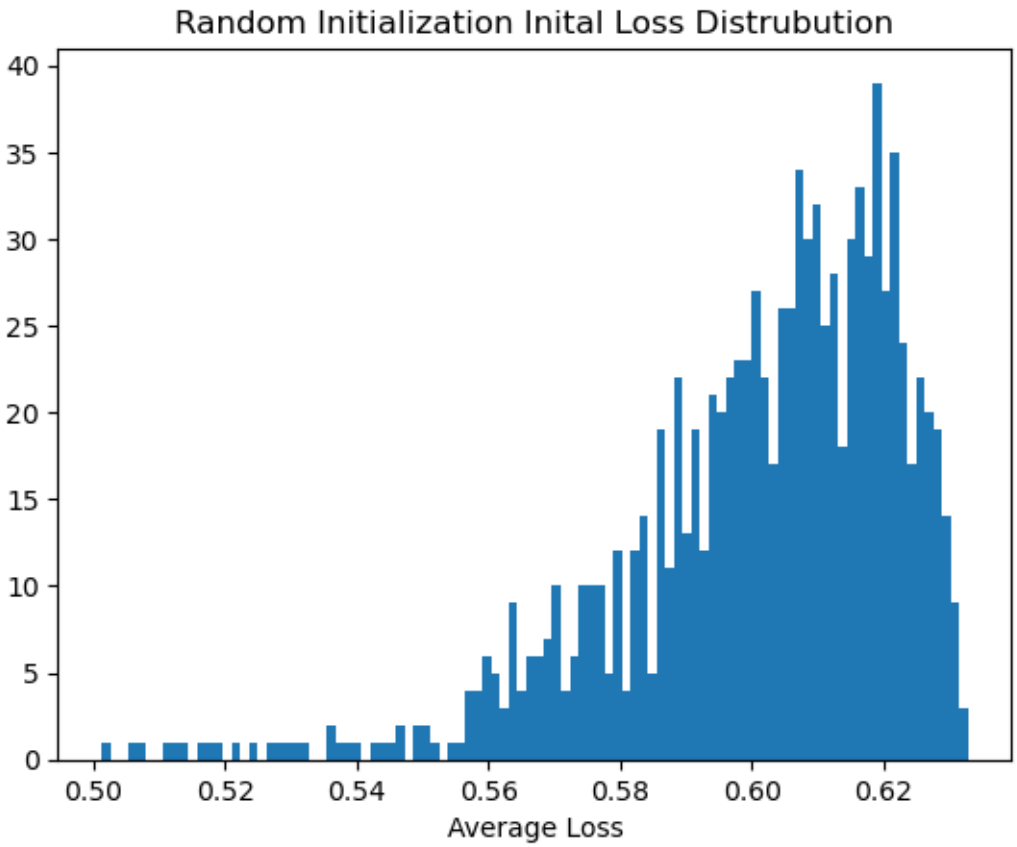
```
                test*=len(np.where(y==i)[0])#testing for empty group
            if test==0:
                while test==0:
                    y=np.random.randint(0,self.C,len(self.data['y']))
                    test=1
                    for i in range(self.C):
                        test*=len(np.where(y==i)[0])
            self.data['y']=y
    def Forgy(self):
        self.mu=[]
        sample=[]
        while len(np.unique(sample))!=self.C:
            sample=np.random.randint(0,len(self.data),self.C) #picking random␣
↪samples to be the means
        for a in sample:
            muk=list(self.data['col1','col2'][a].values()) #apending the␣
↪smaples into the list
            self.mu.append(muk)
    def Kpp(self):
        self.mu=[]
        self.mu.append(self.data['col1','col2'][np.random.randint(0,len(self.
↪data))])# picking one random sample to be fit average
        while len(self.mu) < self.C:
            prob=np.zeros(len(self.data)) #creating empty array for proabilities
            for i in range(len(self.data)):
                t=[0]*len(self.mu) #creating empty array for distnace to each␣
↪average
                for k in range(len(self.mu)):
                    t[k]=self.Dist2(self.data['col1','col2'][i],self.mu[k])**2␣
↪# calculating distnaces
                prob[i]=min(t) # picking smallest distnce for label
            prob/=sum(prob)#nomalizing poablities to sum to 1
            s=np.random.choice(np.arange(0,len(prob),1),p=prob) #picking ranom␣
↪point with probliites of disnaces
            self.mu.append(self.data['col1','col2'][s])#appending new mean
    def Dist2(self,x,y):
        d=0#definding d
        if len(x)==len(y): #chicking for equal length
            for z in range(len(y)):
                d+=(x[z]-y[z])**2 #claulatnig each distnace step
        return np.sqrt(d)
    def cluster(self):
        for i in range(len(self.data)):
            t=[0]*self.C
            for k in range(self.C):
```

```
                t[k]=self.Dist2(self.data['col1','col2'][i],self.mu[k])**2␣
↪#calculating distances
            self.data[i]['y']=np.argmin(t) #labeling with closest mean
    def mean(self):
        for c in range(self.C):
            z=np.where(self.data['y']==c) #finding range for each label
            muk=[np.mean(self.data[z]['col1']),np.mean(self.data[z]['col2'])]␣
↪#finding mean
            self.mu[c]=muk
    def Loss(self,norm=False):
        E=0
        for c in range(self.C):
            z=np.where(self.data['y']==c)
            for i in z[0]:
                E+=self.Dist2(self.data['col1','col2'][i],self.
↪mu[c])#calculating error for each datapoint
        E/=len(self.data) #taking avearge
        if norm:
            return E/np.trace(self.Cov) #normalizing
        else:
            return(E)
    def plot(self):
        for i in range(len(self.data)):
            plt.scatter(self.data[i]['col1'],self.data[i]['col2'],color=self.
↪colors[self.data[i]['y']]) #plotting points
        for c in range(self.C):
            plt.scatter(self.mu[c][0],self.mu[c][1],marker='*',color=self.
↪colors[c],edgecolors='y') #plotting averages
        plt.show()
    def run(self,itt=100):
        for i in range(itt):
            self.oldmu=self.mu.copy() #copying mu
            self.cluster()# asigning labels
            self.mean()#taking new mean
            if self.oldmu==self.mu: #convergnce clause
                print('Convergence Reached',i,' Itterations')
                break
        if i ==itt-1:
            print('Max Itteration Reached') #itteratioin clause
```

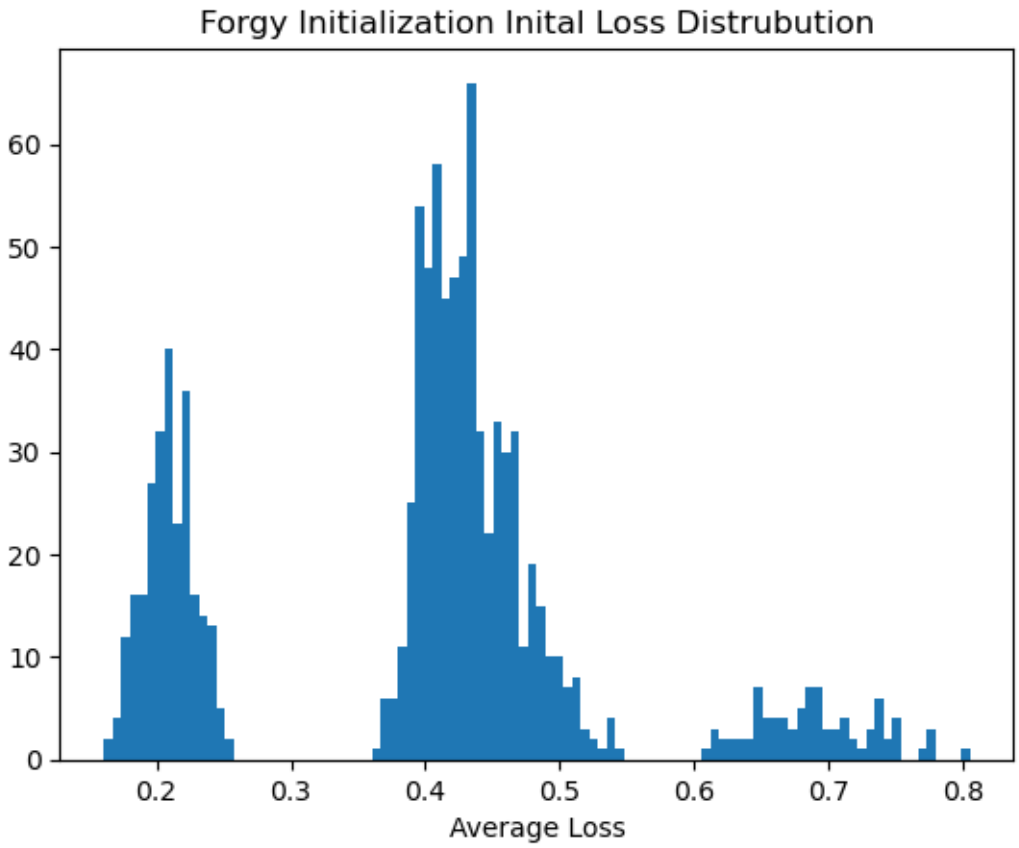
[5]:
```
three=ascii.read('three.csv') # reading in Three dataset
```

[6]:
```
randloss=[]#emty array for losses
for i in range(1000):   #itterating 1000 times
    K=Kmean(three,init='rand') #initilaizing k means with random
    randloss.append(K.Loss()) #appendeing loss into loss array
```

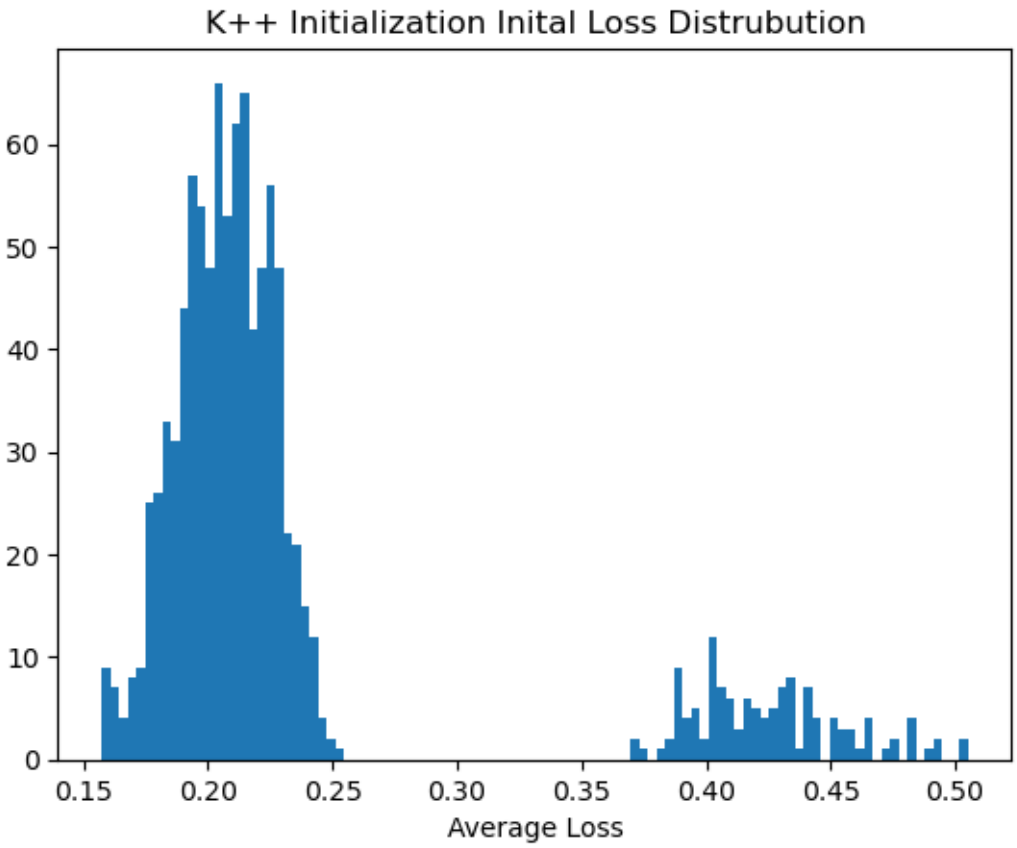
```
#plotting histrogram
plt.title('Random Initialization Inital Loss Distrubution')
plt.xlabel('Average Loss')
plt.hist(randloss,100)
plt.savefig('Images/Random.png',dpi=300)

plt.show()
```



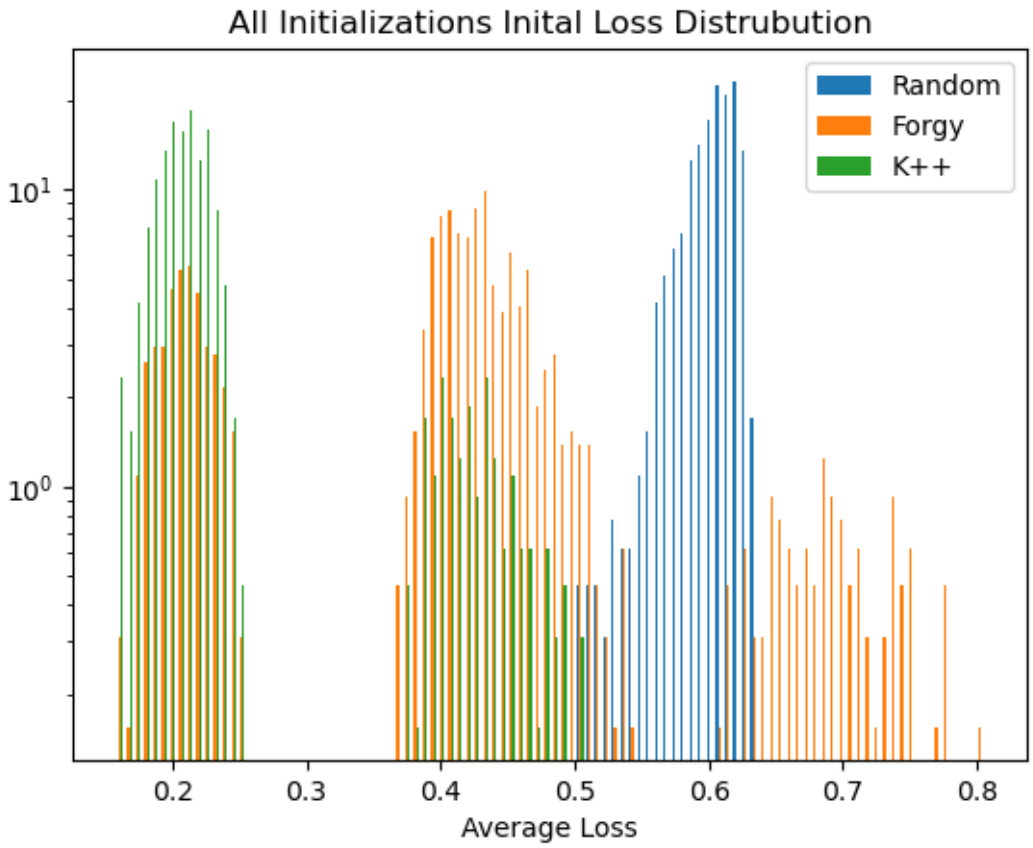
[7]:
```
forgyloss=[]#empty array for losses
for i in range(1000):
    K=Kmean(three,init='forgy') #initaliaing this time with forgy
    forgyloss.append(K.Loss())
plt.title('Forgy Initialization Inital Loss Distrubution')
plt.xlabel('Average Loss')
plt.hist(forgyloss,100)
plt.savefig('Images/Forgy.png',dpi=300)
plt.show()
```



[8]:
```
kpploss=[]
for i in range(1000):
    K=Kmean(three,init='K++') #initaliazing with K++
    kpploss.append(K.Loss())
plt.title('K++ Initialization Inital Loss Distrubution')
plt.xlabel('Average Loss')
plt.hist(kpploss,100)
plt.savefig('Images/K++.png',dpi=300)
plt.show()
```



[9]:
```
plt.
  ↪hist((randloss,forgyloss,kppploss),100,density=True,log=True,label=['Random','Forgy','K++'])
plt.title('All Initializations Inital Loss Distrubution')
plt.xlabel('Average Loss')
plt.legend()
plt.savefig('Images/All.png',dpi=300)
plt.show()
```


All Initializations Inital Loss Distrubution
Random
Forgy
K++
10^1
10^0
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Average Loss

[ ]:

Task3

December 9, 2022

[2]:
```
import matplotlib.pyplot as plt
from astropy.io import ascii
import random
import numpy as np
import math
import copy
%matplotlib inline
```

[3]:
```
class Kmean:
    def __init__(self,data,init='rand',C=3,verb=1):
        self.data=data
        self.verb=verb
        self.init=init
        self.C=C
        self.data['y']=-1
        self.colors=['r','g','b','cyan','magenta','purple','k']
        if self.C>len(self.colors):
            self.colors=self.colors*int(np.ceil(self.C/len(self.
 ↪colors)))#clause for lots of classes
        self.Cov=np.cov(list(self.data.values())) #calcualting covarenince
        if self.init=='rand':
            self.Random()
            self.mean()
        elif self.init=='forgy':
            self.Forgy()
        elif self.init=='K++':
            self.Kpp()
        else:
            print('Initialization type not recognized defaulting to random')
            self.init='rand'
            self.Random()
            self.mean()
        self.cluster()
    def Random(self): #Random init
            self.mu=[[-1,-1]]*self.C #allocating an empy array
            y=np.random.randint(0,self.C,len(self.data['y'])) #assigning␣
 ↪ranomly labels
```

```
            test=1
            for i in range(self.C):
                test*=len(np.where(y==i)[0])#testing for empty group
            if test==0:
                while test==0:
                    y=np.random.randint(0,self.C,len(self.data['y']))
                    test=1
                    for i in range(self.C):
                        test*=len(np.where(y==i)[0])
            self.data['y']=y
    def Forgy(self):
        self.mu=[]
        sample=[]
        while len(np.unique(sample))!=self.C:
            sample=np.random.randint(0,len(self.data),self.C) #picking random␣
↪samples to be the means
        for a in sample:
            muk=list(self.data['col1','col2'][a].values()) #apending the␣
↪smaples into the list
            self.mu.append(muk)
    def Kpp(self):
        self.mu=[]
        self.mu.append(self.data['col1','col2'][np.random.randint(0,len(self.
↪data))])# picking one random sample to be fit average
        while len(self.mu) < self.C:
            prob=np.zeros(len(self.data)) #creating empty array for proabilities
            for i in range(len(self.data)):
                t=[0]*len(self.mu) #creating empty array for distnace to each␣
↪average
                for k in range(len(self.mu)):
                    t[k]=self.Dist2(self.data['col1','col2'][i],self.mu[k])**2␣
↪# claulating distnaces
                prob[i]=min(t) # picking smallest distnce for label
            prob/=sum(prob)#nomalizing poablities to sum to 1
            s=np.random.choice(np.arange(0,len(prob),1),p=prob) #picking ranom␣
↪point with probliites of disnaces
            self.mu.append(self.data['col1','col2'][s])#appending new mean
    def Dist2(self,x,y):
        d=0
        if len(x)==len(y): #chicking for equal length
            for z in range(len(y)):
                d+=(x[z]-y[z])**2 #claulatnig each distnace step
        return np.sqrt(d)
    def cluster(self):
        for i in range(len(self.data)):
            t=[0]*self.C
```

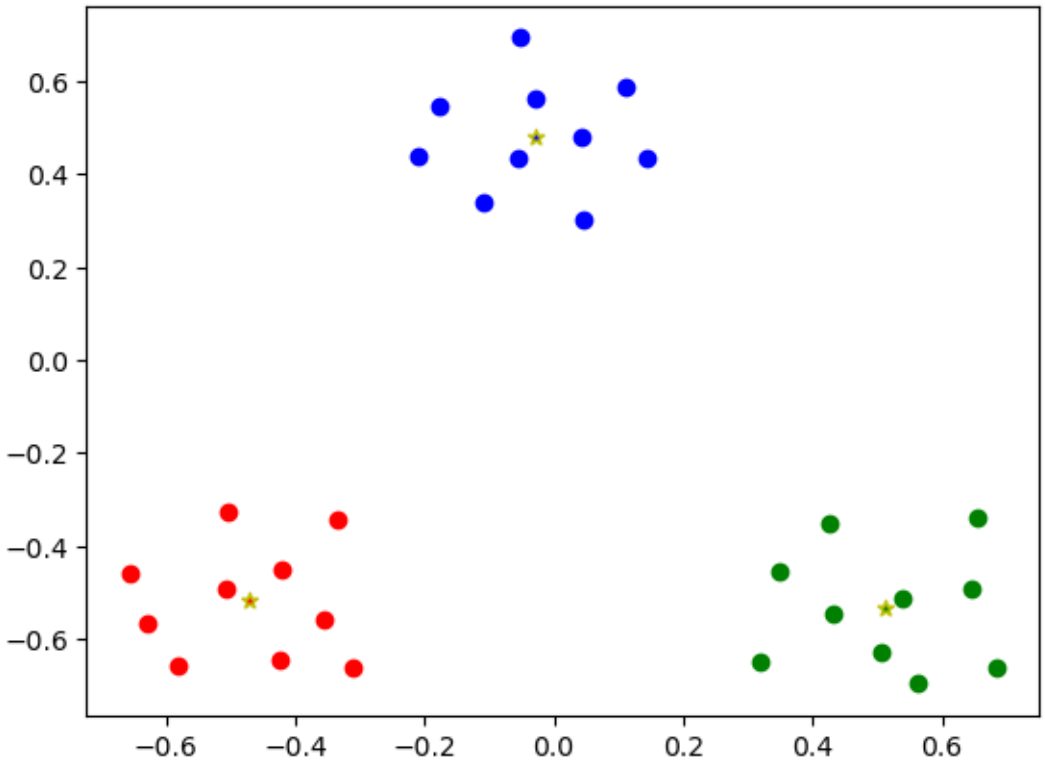
```
            for k in range(self.C):
                t[k]=self.Dist2(self.data['col1','col2'][i],self.mu[k])**2␣
↪#calculating distances
            self.data[i]['y']=np.argmin(t) #labeling with closest mean
    def mean(self):
        for c in range(self.C):
            z=np.where(self.data['y']==c) #finding range for each label
            muk=[np.mean(self.data[z]['col1']),np.mean(self.data[z]['col2'])]␣
↪#finding mean
            self.mu[c]=muk
    def Loss(self,norm=False):
        E=0
        for c in range(self.C):
            z=np.where(self.data['y']==c)
            for i in z[0]:
                E+=self.Dist2(self.data['col1','col2'][i],self.
↪mu[c])#calculating error for each datapoint
        E/=len(self.data) #taking avearge
        if norm:
            return E/np.trace(self.Cov) #normalizing
        else:
            return(E)
    def plot(self,save=False):
        for i in range(len(self.data)):
            plt.scatter(self.data[i]['col1'],self.data[i]['col2'],color=self.
↪colors[self.data[i]['y']]) #plotting points
        for c in range(self.C):
            plt.scatter(self.mu[c][0],self.mu[c][1],marker='*',color=self.
↪colors[c],edgecolors='y') #plotting averages
        if save: #saving it i want to to save
            Name=input('File Name')
            plt.savefig('Images/'+Name,dpi=300)
        plt.show()
    def Sillouette(self,i):
        a=0
        b=0
        z=np.where(self.data[:]['y']==self.data[i]['y'])
        Cs=[0]*self.C
        Csl=[0]*self.C
        for j in range(len(self.data)):
                Cs[int(self.data[j]['y'])]+=self.Dist2(self.
↪data[i]['col1','col2'],self.data[j]['col1','col2'])
                Csl[int(self.data[j]['y'])]+=1
        a=Cs[int(self.data[i]['y'])]
        a/=(len(z[0])-1) #normaling by samples
        B=[Cs[k]/Csl[k] for k in range(len(Cs)) if k!=(int(self.data[i]['y']))]
```

```
        if len(B)!=0:
            b=np.min(B) #normalizing
        else:
            b=a
        #print(a,b)
        #differnt cases for calualting silloute
        if a<b:
            return 1-(a/b)
        elif a>b:
            return (b/a)-1
        else:
            return 0
    def AverageSillouette(self):
        AS=0
        for i in range(len(self.data)):#itterating over all samples
            AS+=self.Sillouette(i)
        return AS/len(self.data) #averageing
    def run(self,itt=100):
        for i in range(itt):
            if self.verb>1: #printting update
                print('Itteration',i+1,'Log10(Loss): ',np.log10(self.
↪Loss(norm=True)))
            self.oldmu = copy.deepcopy(self.mu)# copying mu
            self.mean() #taking new average
            self.cluster() #clustering to saaing label
            if np.array_equal(self.oldmu,self.mu):
                if self.verb>0:
                    print('Convergence Reached',i,' Itterations \nLog10(Loss):
↪',np.log10(self.Loss(norm=True)))
                break
            if i ==itt-1:
                print('Max Itteration Reached')
```

[4]:
```
three=ascii.read('three.csv') #readin data
```

[5]:
```
K=Kmean(three,init='K++',C=3,verb=2)#initlaizing
K.run() #running
K.plot(save=0) #plotting
```

```
Itteration 1 Log10(Loss):  -0.2936110903079981
Itteration 2 Log10(Loss):  -0.4450246881434601
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
```



[6]:
```
Sillouttes=[]#emplty list for sillouttes
```

0.0.1 C=1

[7]:
```
Lossesi=[]# losses for this itteration
Sillouettesi=[] #sillouttes for these intterations
for i in range(10): #doing it 10 times
    print('Itteration',i+1)
    K=Kmean(three,init='K++',C=1)# initialzing
    K.run() #running k menas
    Lossesi.append(np.log10(K.Loss(norm=True)))#apensing losses
    Sillouettesi.append(K.AverageSillouette()) #appedning average silloutte
Sillouttes.append(Sillouettesi[np.argmin(Lossesi)]) #appedning sillopute with␣
 ↪lowest loss
```

```
Itteration 1
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 2
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 3
```

```
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 4
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 5
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 6
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 7
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 8
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 9
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
Itteration 10
Convergence Reached 1  Itterations
Log10(Loss): 0.16882476172207025
```

0.0.2 C=2

[8]:
```
Lossesi=[]# losses for this itteration
Sillouettesi=[] #sillouttes for these intterations
for i in range(10): #doing it 10 times
    print('Itteration',i+1)
    K=Kmean(three,init='K++',C=2)# initialzing
    K.run() #running k menas
    Lossesi.append(np.log10(K.Loss(norm=True)))#apensing losses
    Sillouettesi.append(K.AverageSillouette()) #appedning average silloutte
Sillouttes.append(Sillouettesi[np.argmin(Lossesi)]) #appedning sillopute with␣
 ↪lowest loss
```

```
Itteration 1
Convergence Reached 2  Itterations
Log10(Loss): -0.044405144388677494
Itteration 2
Convergence Reached 3  Itterations
Log10(Loss): -0.04440514438867738
Itteration 3
Convergence Reached 2  Itterations
Log10(Loss): -0.044405144388677494
Itteration 4
Convergence Reached 2  Itterations
```

```
Log10(Loss): -0.003862267066734318
Itteration 5
Convergence Reached 3  Itterations
Log10(Loss): -0.04440514438867738
Itteration 6
Convergence Reached 2  Itterations
Log10(Loss): 0.014709408114682512
Itteration 7
Convergence Reached 2  Itterations
Log10(Loss): -0.044405144388677494
Itteration 8
Convergence Reached 2  Itterations
Log10(Loss): -0.044405144388677494
Itteration 9
Convergence Reached 1  Itterations
Log10(Loss): -0.04440514438867738
Itteration 10
Convergence Reached 2  Itterations
Log10(Loss): -0.044405144388677494
```

0.0.3 C=3

[9]:
```
Lossesi=[]# losses for this itteration
Sillouettesi=[] #silloutte for these intterations
for i in range(10): #doing it 10 times
    print('Itteration',i+1)
    K=Kmean(three,init='K++',C=3)# initialzing
    K.run() #running k menas
    Lossesi.append(np.log10(K.Loss(norm=True)))#apensing losses
    Sillouettesi.append(K.AverageSillouette()) #appedning average silloutte
Sillouttes.append(Sillouettesi[np.argmin(Lossesi)]) #appedning sillopute with␣
 ↪lowest loss
```

```
Itteration 1
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
Itteration 2
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
Itteration 3
Convergence Reached 1  Itterations
Log10(Loss): -0.44502468814346
Itteration 4
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
Itteration 5
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
```

```
Itteration 6
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
Itteration 7
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
Itteration 8
Convergence Reached 2  Itterations
Log10(Loss): -0.44502468814346
Itteration 9
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434599
Itteration 10
Convergence Reached 1  Itterations
Log10(Loss): -0.4450246881434601
```

0.0.4 C=4

[10]:
```
Lossesi=[]# losses for this itteration
Sillouettesi=[] #sillouttes for these intterations
for i in range(10): #doing it 10 times
    print('Itteration',i+1)
    K=Kmean(three,init='K++',C=4)# initialzing
    K.run() #running k menas
    Lossesi.append(np.log10(K.Loss(norm=True)))#apensing losses
    Sillouettesi.append(K.AverageSillouette()) #appedning average silloutte
Sillouttes.append(Sillouettesi[np.argmin(Lossesi)]) #appedning sillopute with␣
 ↪lowest loss
```

```
Itteration 1
Convergence Reached 3  Itterations
Log10(Loss): -0.4752254129350596
Itteration 2
Convergence Reached 1  Itterations
Log10(Loss): -0.47662384726962326
Itteration 3
Convergence Reached 1  Itterations
Log10(Loss): -0.47541437441524853
Itteration 4
Convergence Reached 3  Itterations
Log10(Loss): -0.47466319198681883
Itteration 5
Convergence Reached 1  Itterations
Log10(Loss): -0.47982938382063484
Itteration 6
Convergence Reached 1  Itterations
Log10(Loss): -0.48423718745499483
Itteration 7
```

```
Convergence Reached 1  Itterations
Log10(Loss): -0.4701008038612992
Itteration 8
Convergence Reached 1  Itterations
Log10(Loss): -0.47801634741943316
Itteration 9
Convergence Reached 1  Itterations
Log10(Loss): -0.4820282902098736
Itteration 10
Convergence Reached 1  Itterations
Log10(Loss): -0.48126063585948803
```

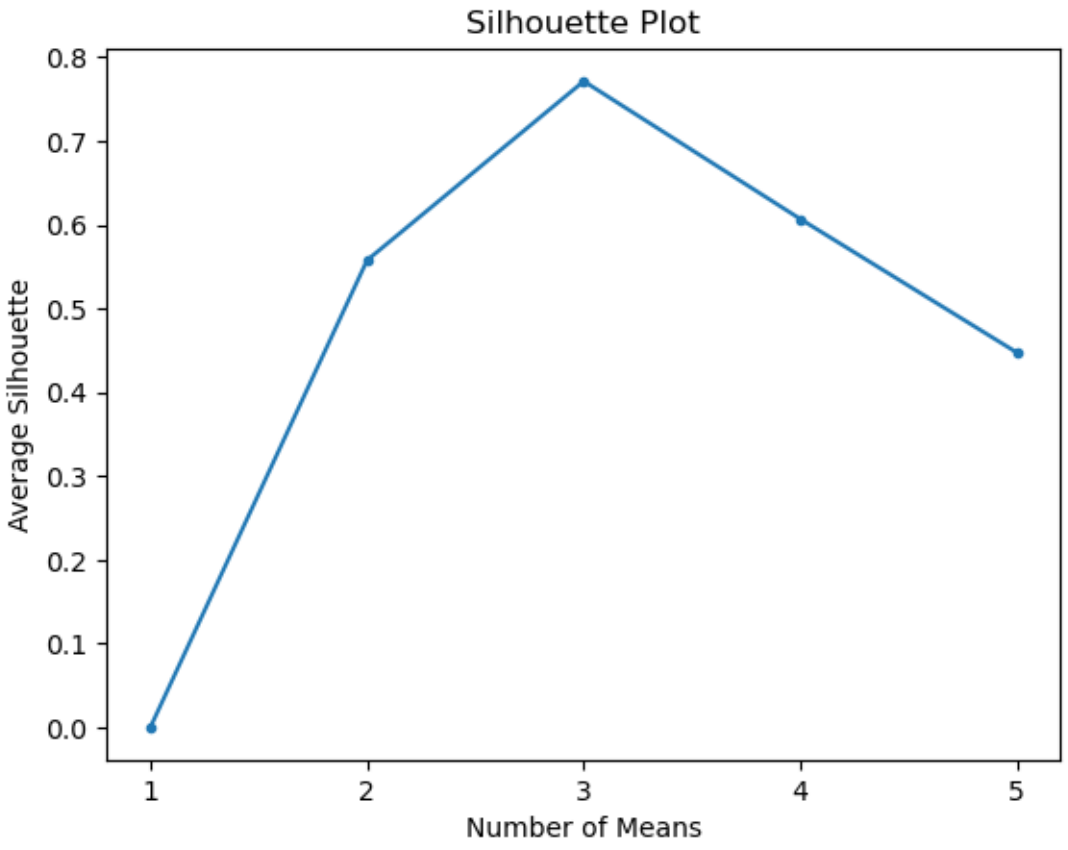
0.0.5 C=5

[11]:
```
Lossesi=[]# losses for this itteration
Sillouettesi=[] #sillouttes for these intterations
for i in range(10): #doing it 10 times
    print('Itteration',i+1)
    K=Kmean(three,init='K++',C=5)# initialzing
    K.run() #running k menas
    Lossesi.append(np.log10(K.Loss(norm=True)))#apensing losses
    Sillouettesi.append(K.AverageSillouette()) #appedning average silloutte
Sillouttes.append(Sillouettesi[np.argmin(Lossesi)]) #appedning sillopute with␣
 ↪lowest loss
```

```
Itteration 1
Convergence Reached 1  Itterations
Log10(Loss): -0.5085133681644615
Itteration 2
Convergence Reached 1  Itterations
Log10(Loss): -0.5069132409177446
Itteration 3
Convergence Reached 2  Itterations
Log10(Loss): -0.5188534023780814
Itteration 4
Convergence Reached 2  Itterations
Log10(Loss): -0.5030074840510153
Itteration 5
Convergence Reached 1  Itterations
Log10(Loss): -0.5216383724478182
Itteration 6
Convergence Reached 2  Itterations
Log10(Loss): -0.5089196859828199
Itteration 7
Convergence Reached 2  Itterations
Log10(Loss): -0.5118339892806153
Itteration 8
Convergence Reached 2  Itterations
```

```
Log10(Loss): -0.5140123667759062
Itteration 9
Convergence Reached 1  Itterations
Log10(Loss): -0.5128563980436938
Itteration 10
Convergence Reached 1  Itterations
Log10(Loss): -0.5169560418858028
```

[14]:
```
#plotting average silloutte
plt.plot([1,2,3,4,5],Sillouttes,marker='.')
plt.xticks([1,2,3,4,5])
plt.xlabel('Number of Means')
plt.ylabel('Average Silhouette')
plt.title('Silhouette Plot')
plt.savefig('Images/SillouttePlot.png',dpi=300)
plt.show()
print(Sillouttes)
```

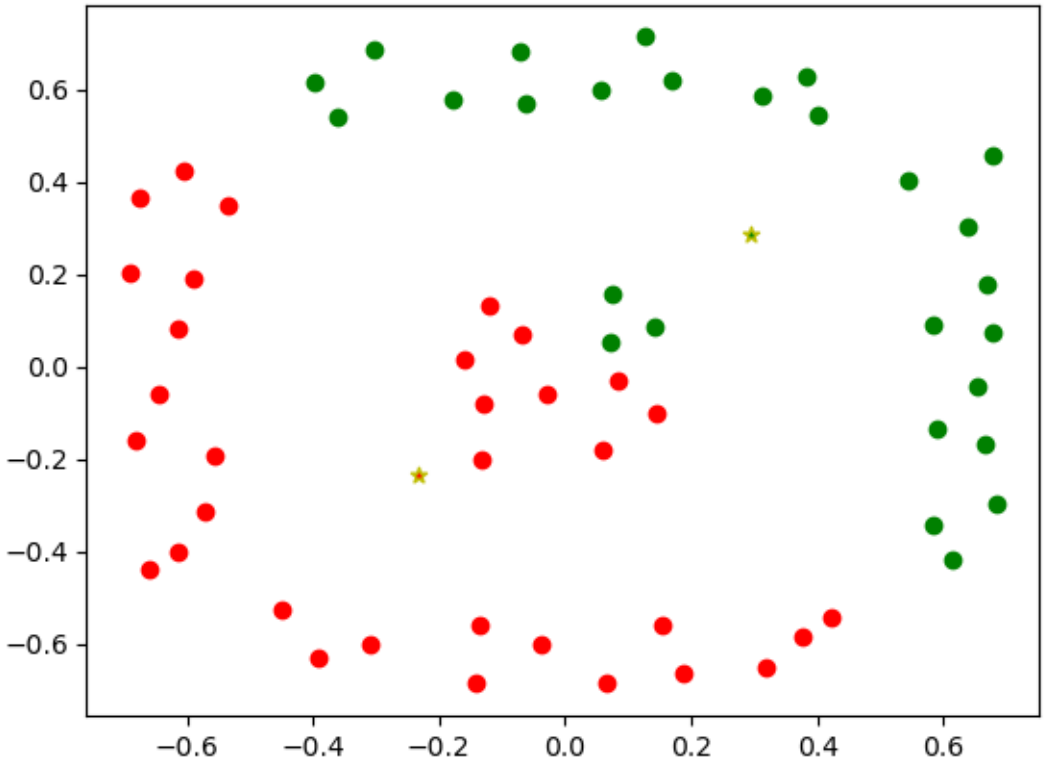



```
[0.0, 0.5575974178219295, 0.7713845060772166, 0.6067461132539519,
0.44716701899097383]
```

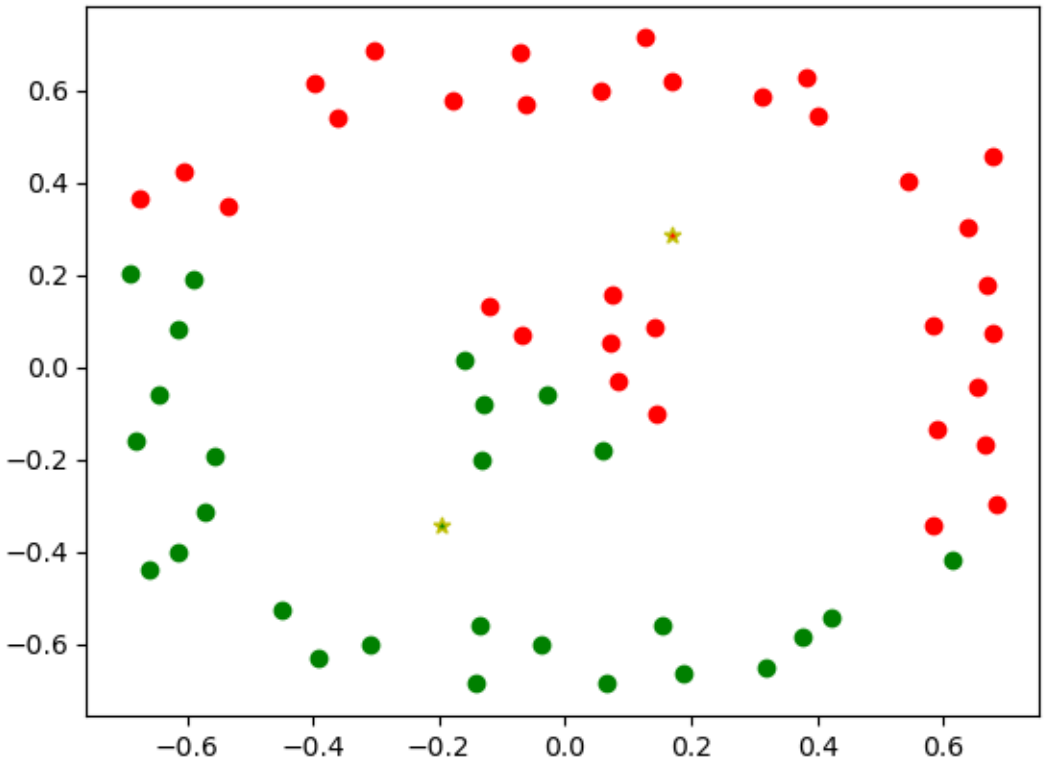
[237]:
```
atoll=ascii.read('atoll.csv') #reading in new dataset
```

[238]:
```
#running k-means with C=2 10 times
for i in range(10):
    print('Itteration',i+1)
    K2=Kmean(atoll,init='K++',C=2,verb=1)
    K2.run()
    K2.plot()
```

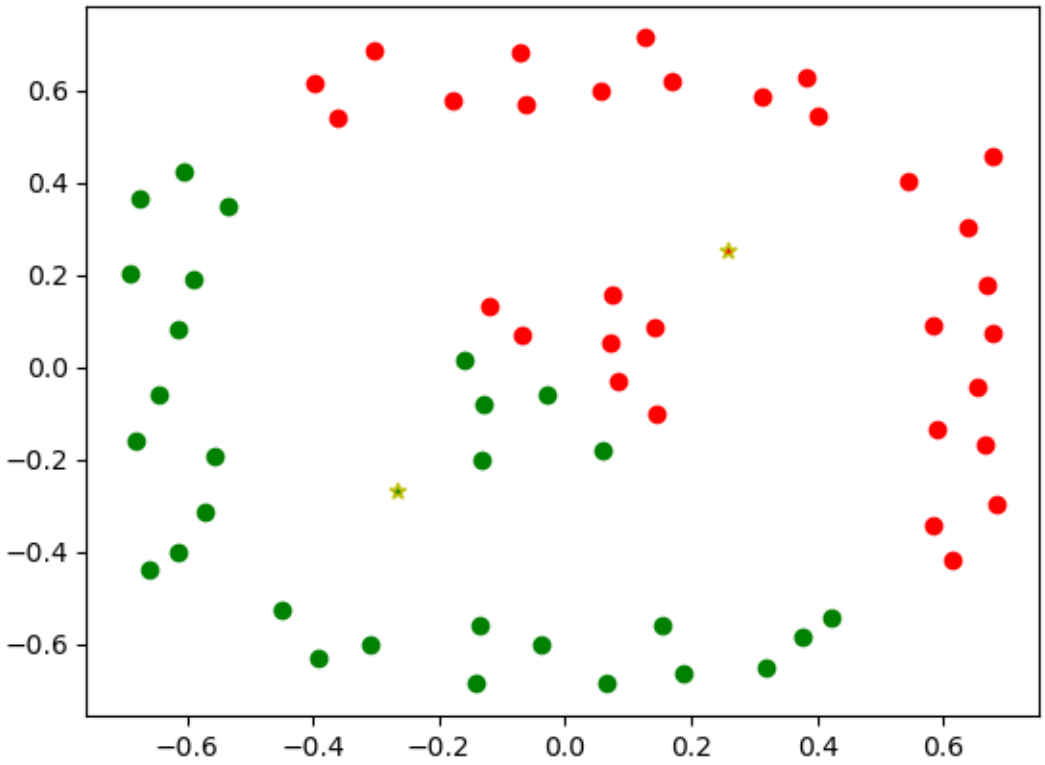
```
Itteration 1
Convergence Reached 6  Itterations
Log10(Loss): 0.08233017340176049
```



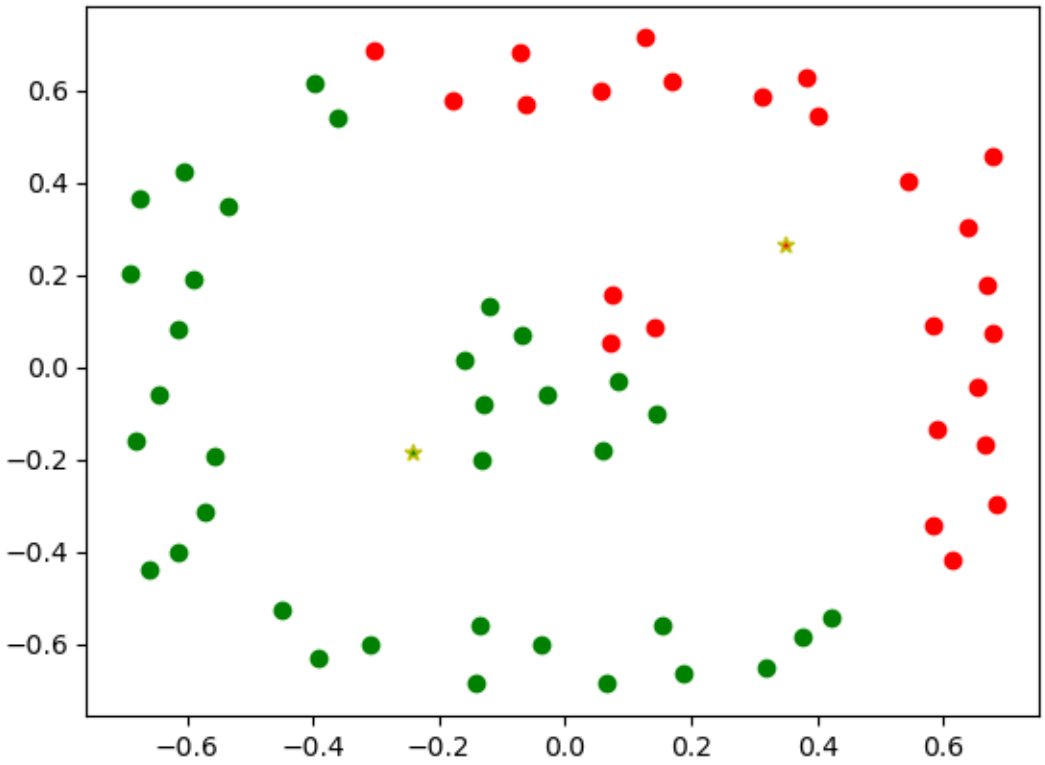

```
Itteration 2
Convergence Reached 2  Itterations
Log10(Loss): 0.08482962788031433
```



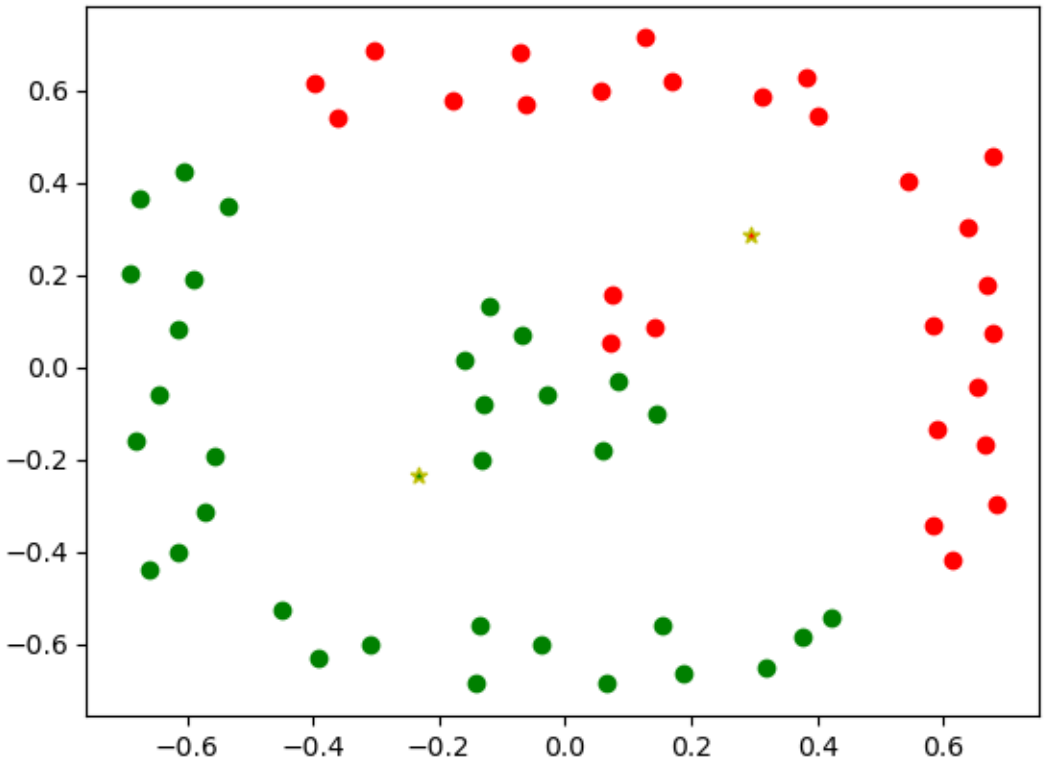

```
Itteration 3
Convergence Reached 2  Itterations
Log10(Loss): 0.08268644619586546
```



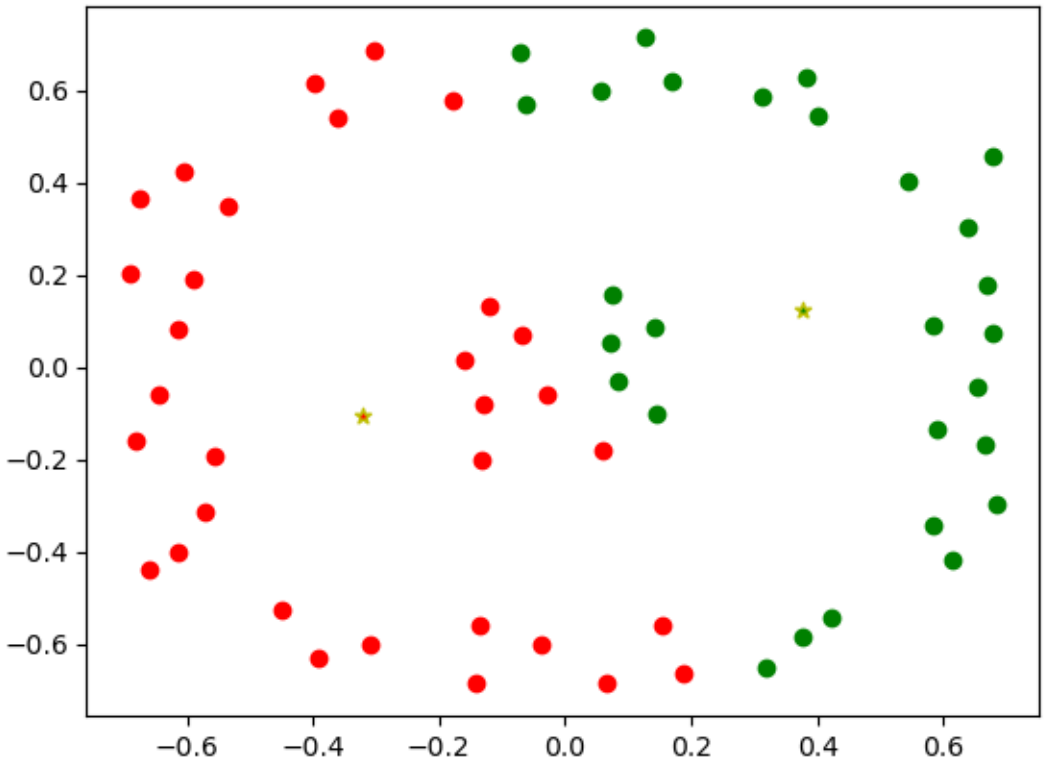

```
Itteration 4
Convergence Reached 1  Itterations
Log10(Loss): 0.0827862693897903
```



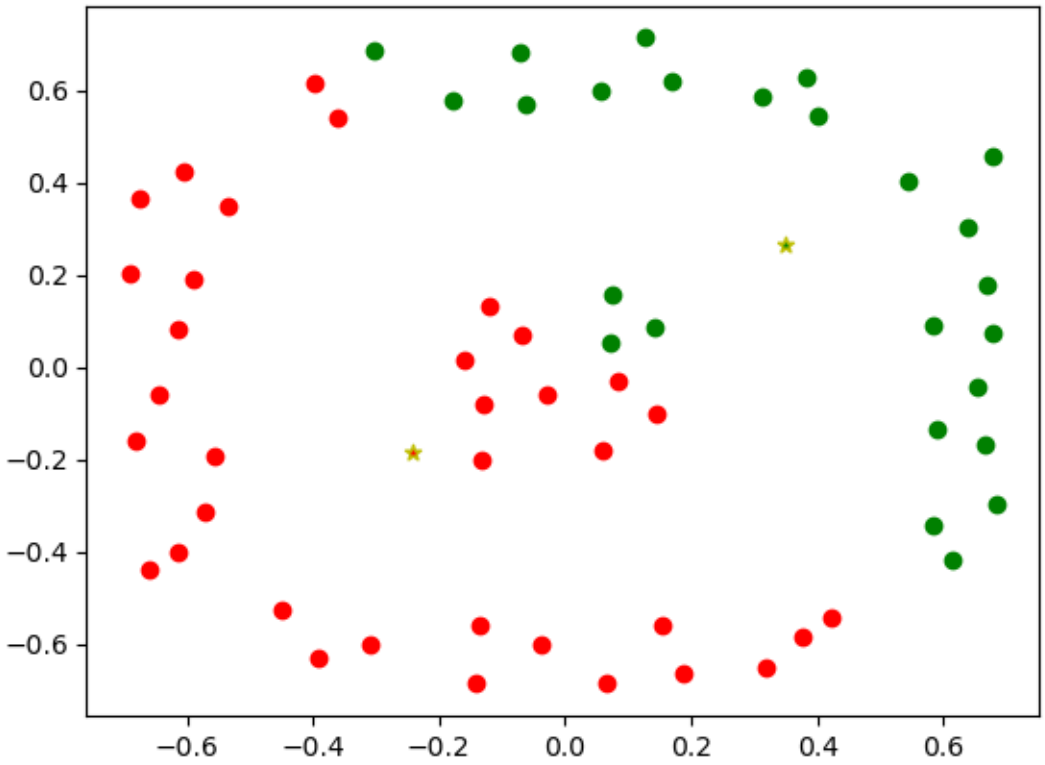

```
Itteration 5
Convergence Reached 3  Itterations
Log10(Loss): 0.08233017340176066
```



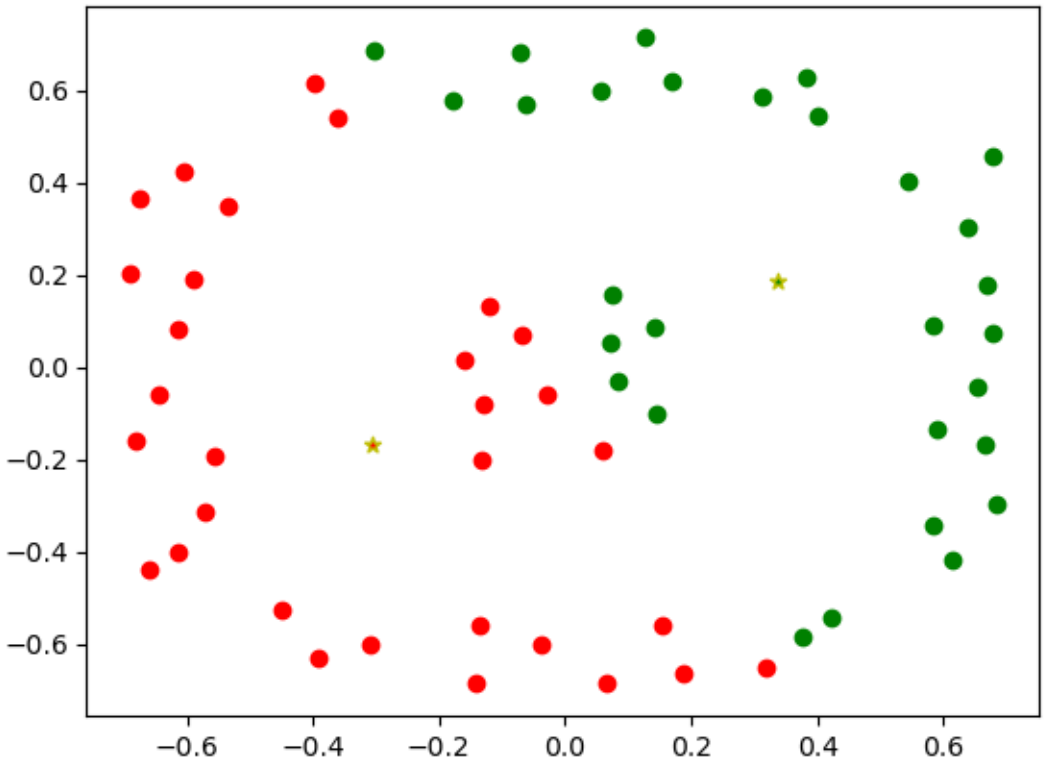

```
Itteration 6
Convergence Reached 1  Itterations
Log10(Loss): 0.08368885266338598
```



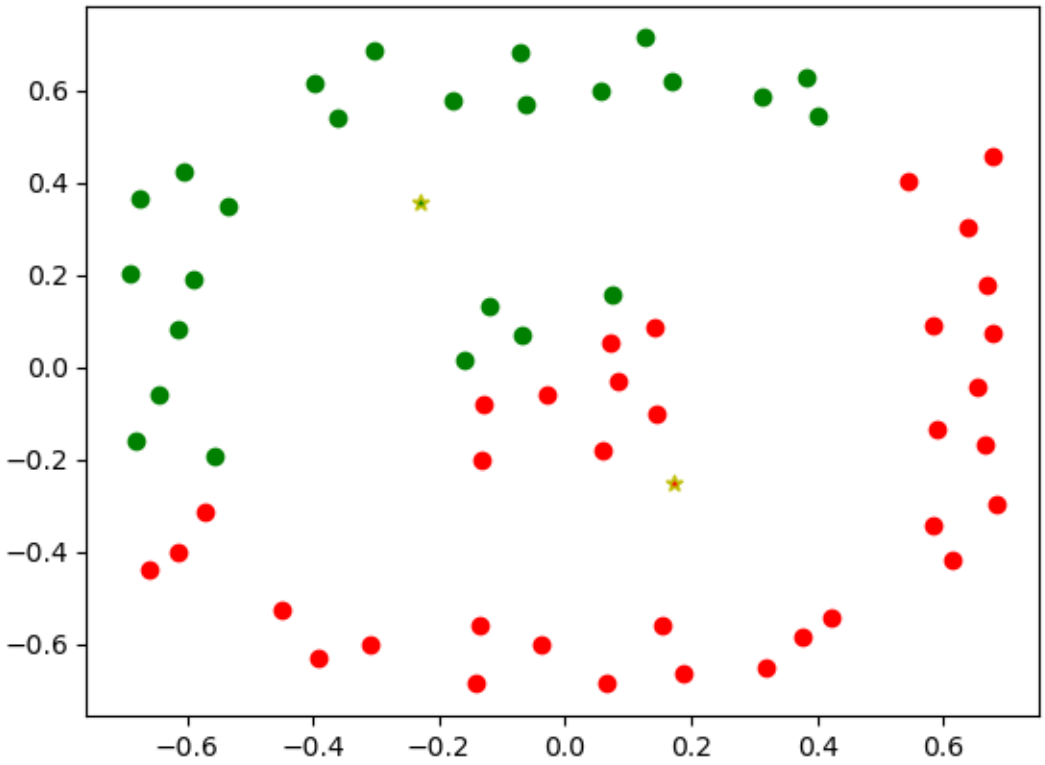

```
Itteration 7
Convergence Reached 5  Itterations
Log10(Loss): 0.0827862693897903
```



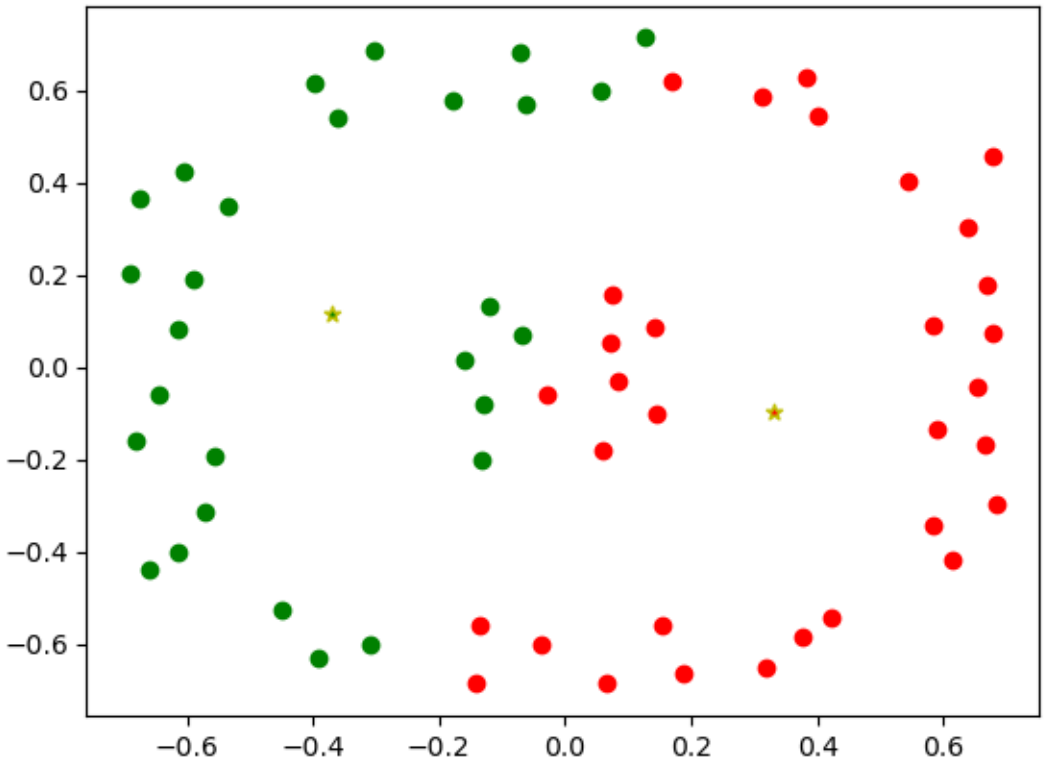

```
Itteration 8
Convergence Reached 1  Itterations
Log10(Loss): 0.08374466471199116
```




```
Itteration 9
Convergence Reached 5  Itterations
Log10(Loss): 0.0847801505867826
```

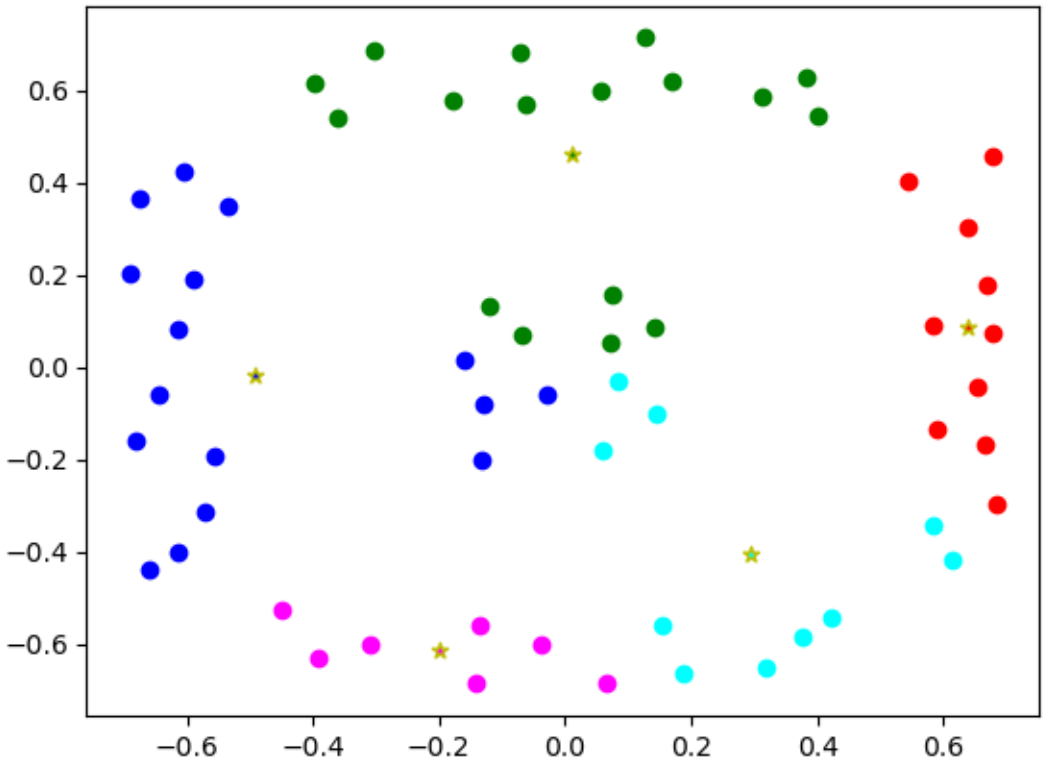



```
Itteration 10
Convergence Reached 5  Itterations
Log10(Loss): 0.08442532022009569
```

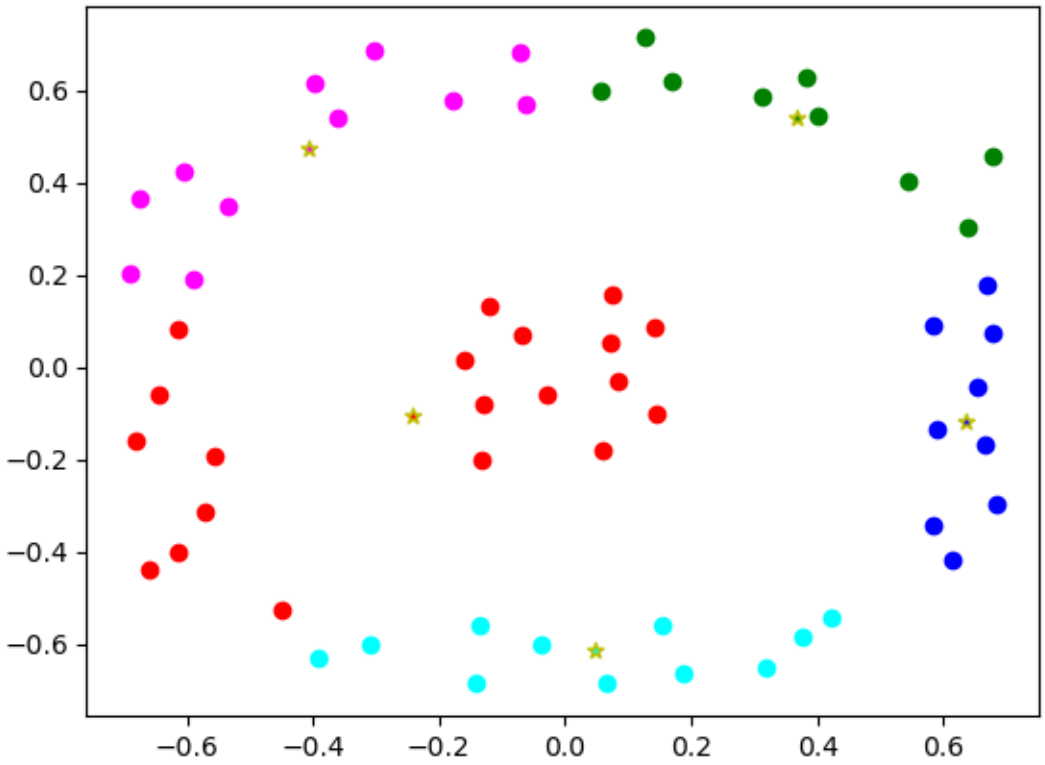


[239]:
```
#running k-means with C=5 10 times
for i in range(10):
    print('Itteration',i+1)
    K2=Kmean(atoll,init='K++',C=5,verb=1)
    K2.run()
    K2.plot()
```

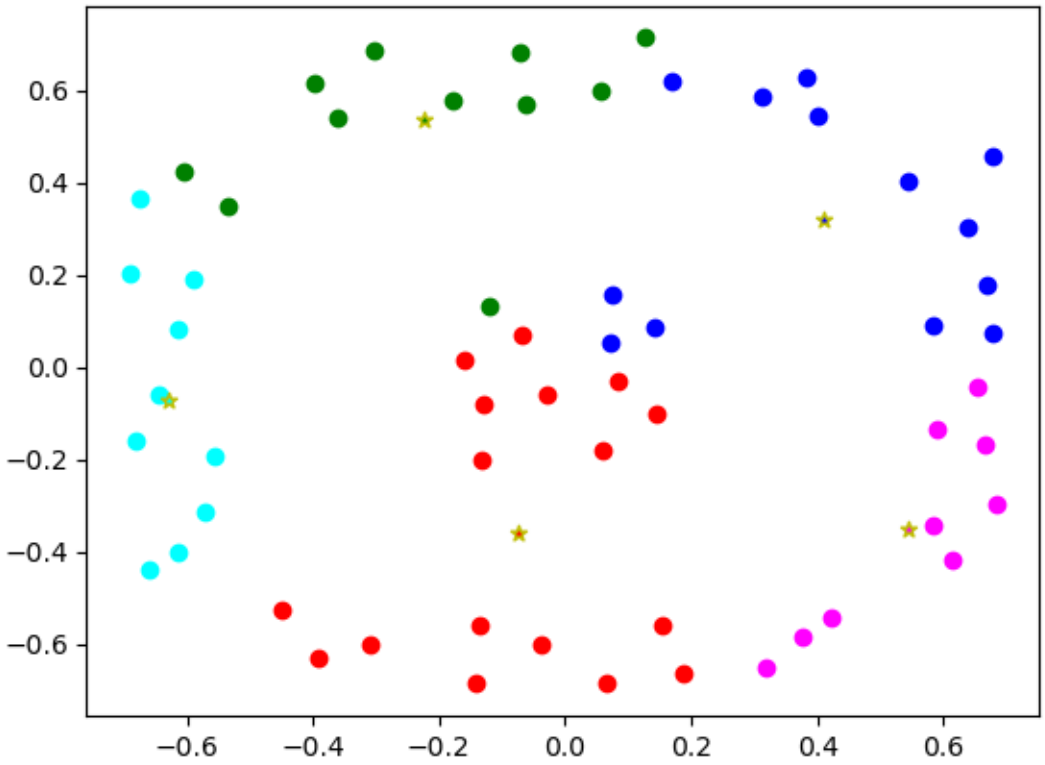
```
Itteration 1
Convergence Reached 3  Itterations
Log10(Loss): -0.13585562433891277
```



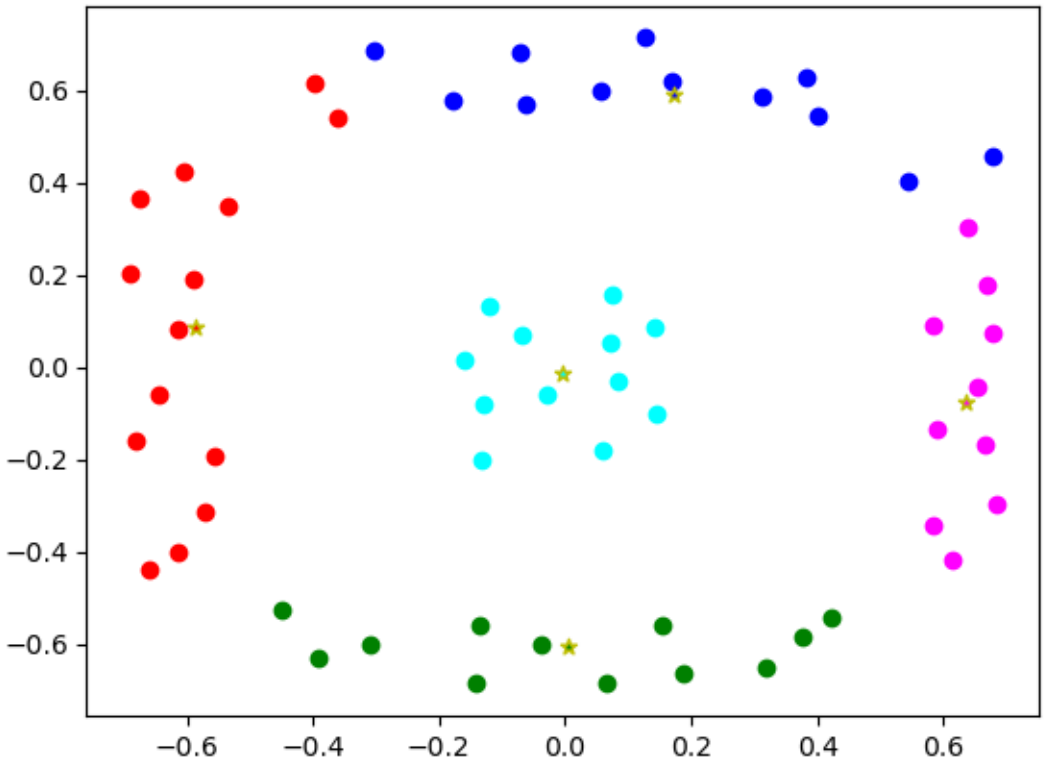

```
Itteration 2
Convergence Reached 2  Itterations
Log10(Loss): -0.16128003029237103
```



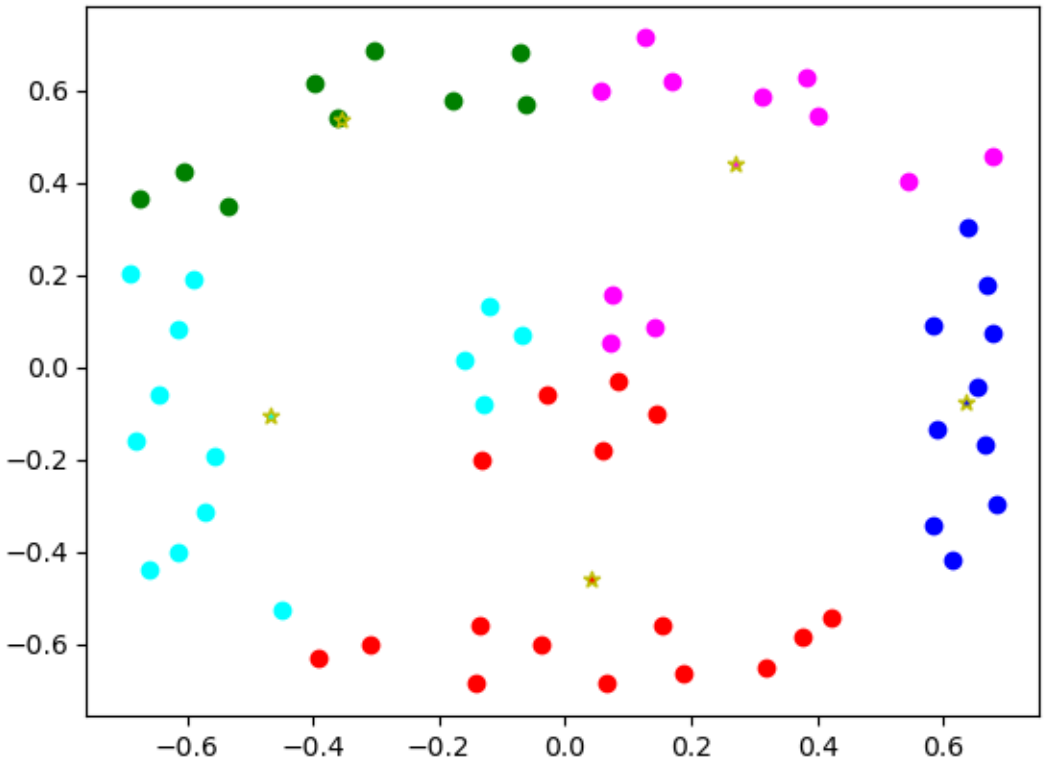

```
Itteration 3
Convergence Reached 3  Itterations
Log10(Loss): -0.141098035455307
```



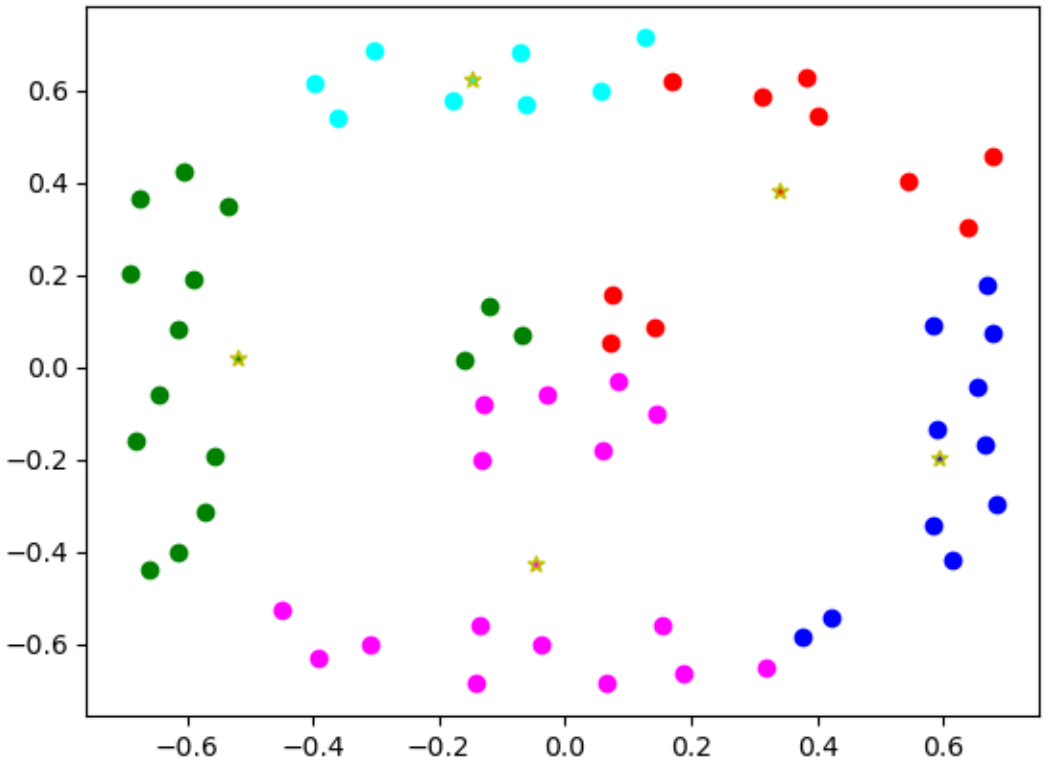

```
Itteration 4
Convergence Reached 3  Itterations
Log10(Loss): -0.20269822997847461
```



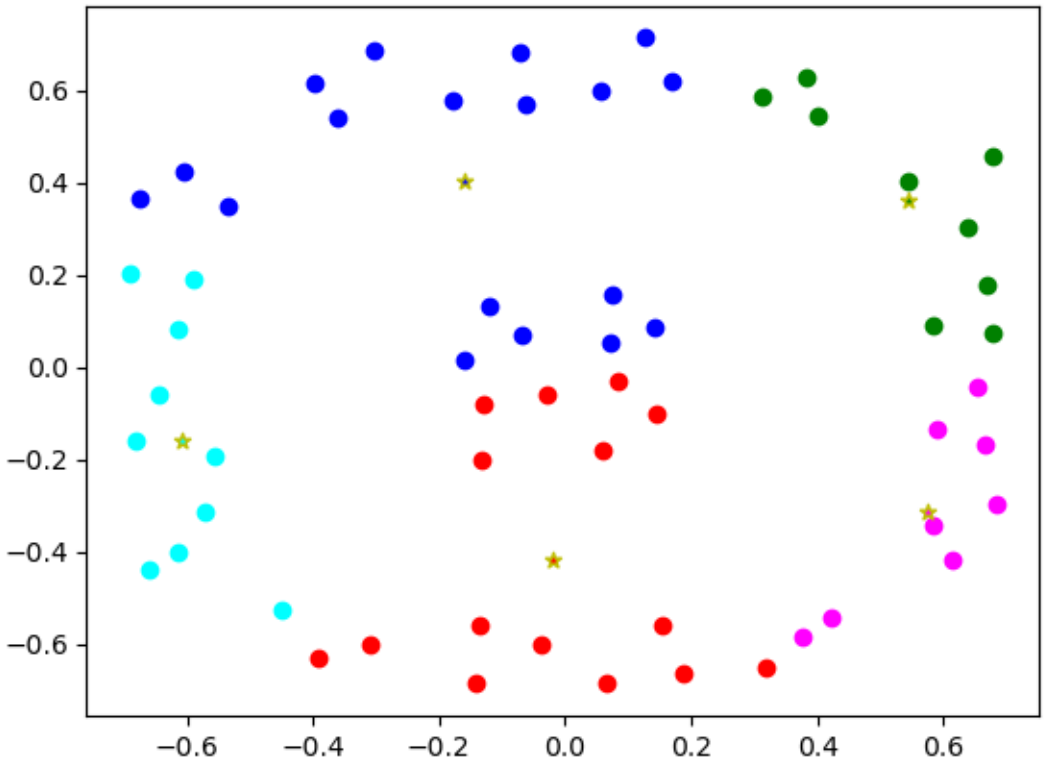

```
Itteration 5
Convergence Reached 5  Itterations
Log10(Loss): -0.14337576295108878
```



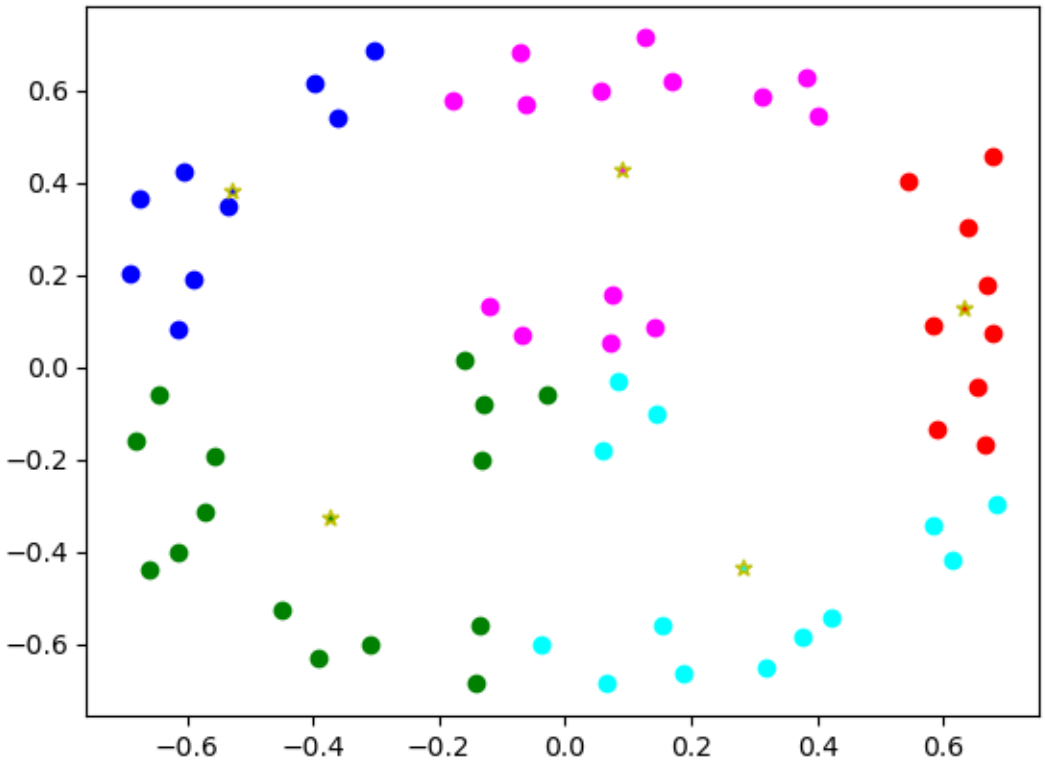

```
Itteration 6
Convergence Reached 3  Itterations
Log10(Loss): -0.1383555081687376
```



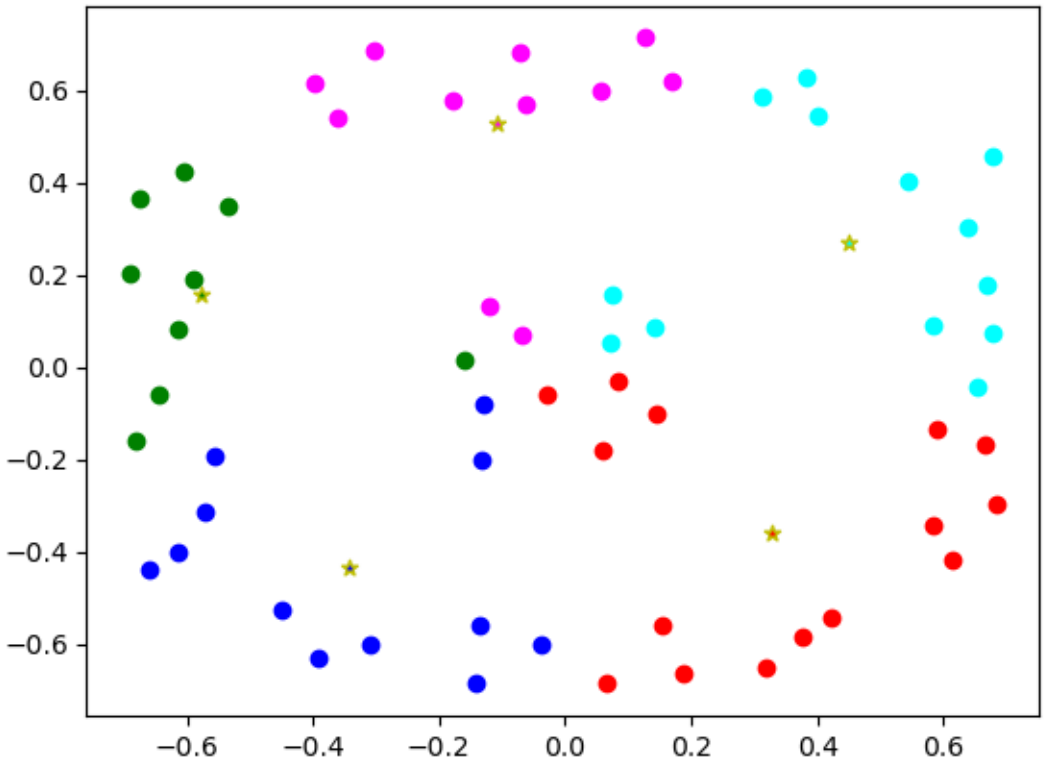

```
Itteration 7
Convergence Reached 4  Itterations
Log10(Loss): -0.14083088991030523
```



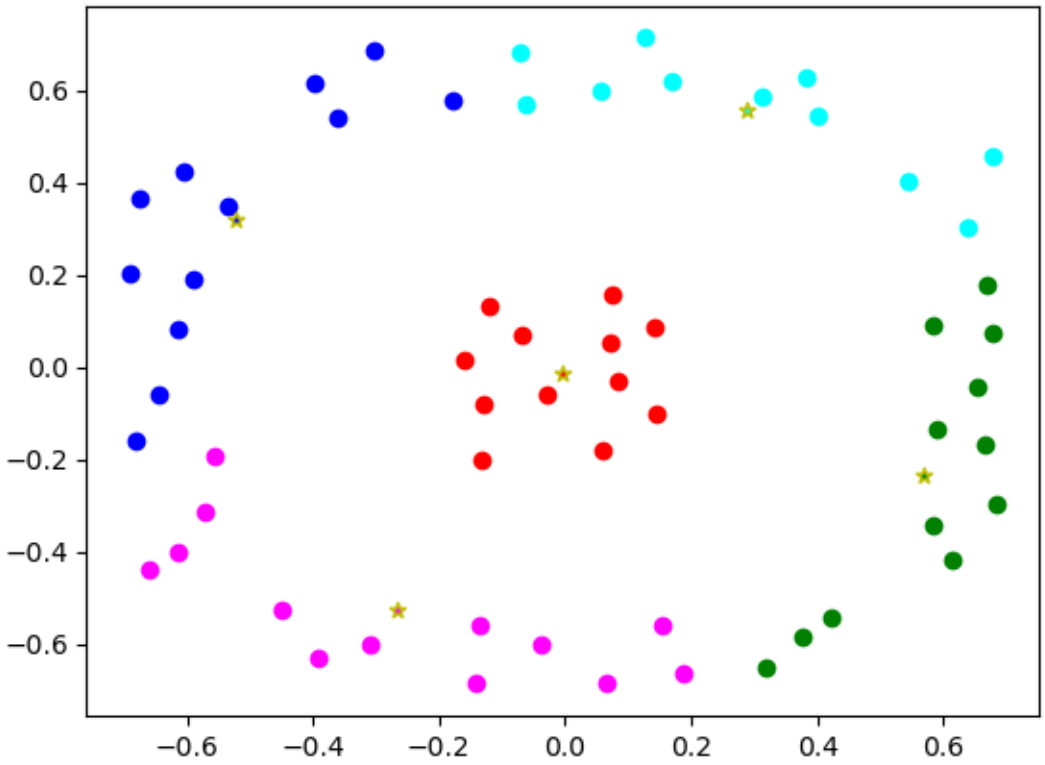

```
Itteration 8
Convergence Reached 6  Itterations
Log10(Loss): -0.14478709438628443
```




```
Itteration 9
Convergence Reached 3  Itterations
Log10(Loss): -0.1344106190411504
```




```
Itteration 10
Convergence Reached 3  Itterations
Log10(Loss): -0.1903069652964721
```


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

[ ]:

Task4

December 9, 2022

[5]:
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
from astropy.io import ascii
from itertools import product
import copy
from datetime import datetime
from skimage.metrics import peak_signal_noise_ratio as PSNR
```

[74]:
```
night=Image.open('Images/vangogh-starry-night.png')# reeading in image
w, h = night.size #saving image size
reduced=np.array(night)/255 #reducing size of pixel values to not get overflow
vecs=[]#empty list of vecotros
#taking chunks and vectorizing them
for i in range(0,reduced.shape[0],5):
    for j in range(0,reduced.shape[1],5):
        Patch=reduced[i:i+5,j:j+5]
        vecs.append(Patch.reshape(25))
vecs=np.array(vecs)
```

[74]:
```
array([[0.05882353, 0.14901961, 0.14117647, …, 0.18823529, 0.14901961,
        0.07843137],
       [0.19607843, 0.23137255, 0.27843137, …, 0.28627451, 0.27058824,
        0.18431373],
       [0.22745098, 0.22745098, 0.54901961, …, 0.45098039, 0.45490196,
        0.27058824],
       …,
       [0.39215686, 0.33333333, 0.3254902 , …, 0.00392157, 0.00392157,
        0.00784314],
       [0.34901961, 0.58431373, 0.67058824, …, 0.00784314, 0.00784314,
        0.00784314],
       [0.79607843, 0.91372549, 0.78823529, …, 0.00784314, 0.00784314,
        0.00784314]])
```

[75]:
```
class KmeanVQ:
    def __init__(self,data,init='rand',C=3):
        #saving init values
```

```
        self.data=data
        self.init=init
        self.C=C
        #making empty array of labels
        self.labels=np.zeros(len(data))
        self.colors=['r','g','b','cyan','magenta','purple','k']
        self.Cov=np.cov(self.data) #calculating
        #inialization cases
        if self.init=='rand':
            self.Random()
            self.mean()
        elif self.init=='forgy':
            self.Forgy()
            self.cluster()
        elif self.init=='K++':
            self.Kpp()
            self.cluster()
        else:
            print('Initialization type not recognized defaulting to random')
            self.init='rand'
            self.Random()
            self.mean()
    def Random(self): #ranomd inilazion case
            self.mu=[[-1,-1]]*self.C
            y=np.random.randint(0,self.C,len(self.labels))
            test=1
            for i in range(self.C):
                test*=len(np.where(y==i)[0])
            if test==0:
                while test==0:
                    y=np.random.randint(0,self.C,len(self.labels))
                    test=1
                    for i in range(self.C):
                        test*=len(np.where(y==i)[0])
            self.labels=y
    def Forgy(self): #forgy initalizaion
        self.mu=[]
        sample=[]
        while len(np.unique(sample))!=self.C:
            sample=np.random.randint(0,len(self.data),self.C)
        for a in sample:
            muk=list(self.data[a].values())
            self.mu.append(muk)
    def Kpp(self): #kmeans ++
        self.mu=np.zeros((self.C,1,25))
        self.mu[0]=self.data[np.random.randint(0,len(self.data))]
        for i in range(1,self.C):
```

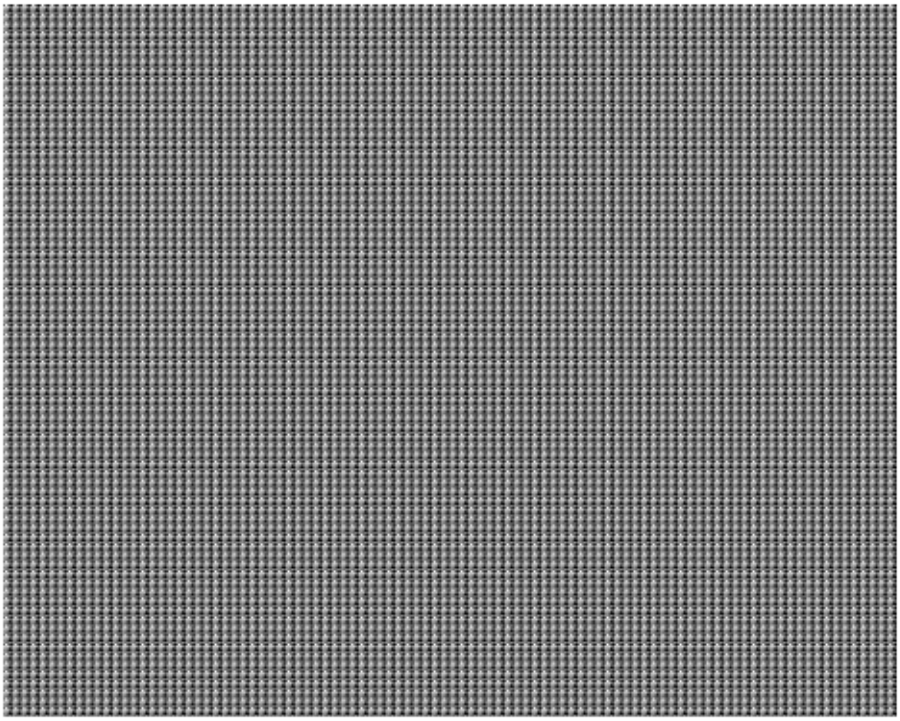
```
            print(i,'Averages Foud',end='\r')#pringing updatges for number of␣
↪averges found
            prob=np.sum((self.mu-self.data)*(self.mu-self.data),axis=2).
↪min(axis=0)
            prob/=sum(prob)
            s=np.random.choice(np.arange(0,len(prob),1),p=prob)
            self.mu[i]=self.data[s]
        print('All Averages Found')
    def Dist2(self,x,y): #calculating distances
        d=np.sum((x-y)**2)
        return d
    def cluster(self):#clustering
        self.labels=np.sum((self.mu-self.data)*(self.mu-self.data),axis=2).
↪argmin(axis=0) #assingin labels
    def mean(self): #finfing averages for classes
        for c in range(self.C):
            z=np.where(self.labels==c)
            muk=[np.mean(self.data[z],axis=0)]
            self.mu[c]=muk
    def Loss(self,norm=False): #claculating losses
        E=0
        for c in range(self.C):
            z=np.where(self.labels==c)
            for i in z[0]:
                E+=self.Dist2(self.data[i],self.mu[c])
        if norm:
            return E/np.trace(self.Cov)
        else:
            return(E)
    def run(self,itt=100):
        for i in range(itt):
            print('Itteration',i,'Time: ', datetime.now().strftime("%H:%M:
↪%S"),'Loss: ',self.Loss(norm=True),end='\r')#running with outputs
            self.oldmu = copy.deepcopy(self.mu)
            self.mean()
            self.cluster()
            #self.plot()
            if np.array_equal(self.oldmu,self.mu):
                print('Convergence Reached',i,' Itterations')
                break
            if i ==itt-1:
                print('Max Itteration Reached')
```

[78]:
```
runs1=[]
losses1=[]
for i in range(10): #runnnig kmeans c=1 10 times
    print(datetime.now().strftime("%H:%M:%S"))
```

```
    K=KmeanVQ(vecs,'K++',C=1)
    K.run(itt=1000)
    runs1.append(K)
    losses1.append(K.Loss(norm=True))
KBest=runs1[np.argmin(losses1)] #picking best run
print('Lowest Loss:',np.min(losses1)) #print lowest loss
comp=[]
#rebuilding image
for i in range(len(vecs)):
    comp.append(KBest.mu[int(KBest.labels[i])].reshape(5,5))
for i in range(0,reduced.shape[0],5):
    for j in range(0,reduced.shape[1],5):
        img[i:i+5,j:j+5]=comp.pop(0)
#calculating PSNR
print('PSNR:',PSNR(np.array(night,dtype='float64'),img*255,data_range=255))
#printing new picture
plt.imshow(img*255,cmap='Greys_r')
plt.axis('off')
plt.savefig("Images/Night1.png",bbox_inches='tight',dpi=300)
```

```
23:29:53
All Averages Found
Convergence Reached 1  Itterations  54.664672747015248
23:29:54
All Averages Found
Convergence Reached 1  Itterations  54.66467274701524
23:29:54
All Averages Found
Convergence Reached 1  Itterations  54.664672747015247
23:29:55
All Averages Found
Convergence Reached 1  Itterations  54.66467274701524
23:29:56
All Averages Found
Convergence Reached 1  Itterations  54.664672747015242
23:29:56
All Averages Found
Convergence Reached 1  Itterations  54.664672747015249
23:29:57
All Averages Found
Convergence Reached 1  Itterations  54.66467274701524
23:29:57
All Averages Found
Convergence Reached 1  Itterations  54.66467274701524
23:29:58
All Averages Found
Convergence Reached 1  Itterations  54.66467274701524
```

```
23:29:58
All Averages Found
Convergence Reached 1  Itterations  54.664672747015248
Lowest Loss: 54.66467274701524
PSNR: 13.854846479605303
```



[80]:
```
# same as prv cell b ut with C=10
runs10=[]
losses10=[]
for i in range(10):
    print(datetime.now().strftime("%H:%M:%S"))
    K=KmeanVQ(vecs,'K++',C=10)
    K.run(itt=1000)
    runs10.append(K)
    losses10.append(K.Loss(norm=True))
KBest=runs10[np.argmin(losses10)]
print('Lowest Loss:',np.min(losses10))
comp=[]
for i in range(len(vecs)):
    comp.append(KBest.mu[int(KBest.labels[i])].reshape(5,5))
for i in range(0,reduced.shape[0],5):
    for j in range(0,reduced.shape[1],5):
        img[i:i+5,j:j+5]=comp.pop(0)
print('PSNR:',PSNR(np.array(night,dtype='float64'),img*255,data_range=255))
```

```
plt.imshow(img*255,cmap='Greys_r')
plt.axis('off')
plt.savefig("Images/Night10.png",bbox_inches='tight',dpi=300)
```

```
23:30:16
All Averages Found
Convergence Reached 45  Itterations  22.766640428154166
23:30:20
All Averages Found
Convergence Reached 47  Itterations  22.722176089262426
23:30:24
All Averages Found
Convergence Reached 61  Itterations  22.740322028883252
23:30:30
All Averages Found
Convergence Reached 93  Itterations  22.723523442307094
23:30:38
All Averages Found
Convergence Reached 54  Itterations  22.827033386949992
23:30:43
All Averages Found
Convergence Reached 64  Itterations  22.737669666005978
23:30:49
All Averages Found
Convergence Reached 73  Itterations  22.866246908466756
23:30:55
All Averages Found
Convergence Reached 46  Itterations  22.787932804068767
23:30:59
All Averages Found
Convergence Reached 82  Itterations  22.736741275302627
23:31:06
All Averages Found
Convergence Reached 73  Itterations  22.744426321813744
Lowest Loss: 22.722176089262426
PSNR: 17.667414793275142
```

[83]:
```
# same as prv cell b ut with C=100
runs100=[]
losses100=[]
for i in range(10):
    print(datetime.now().strftime("%H:%M:%S"))
    K=KmeanVQ(vecs,'K++',C=100)
    K.run(itt=1000)
    runs100.append(K)
    losses100.append(K.Loss(norm=True))
KBest=runs100[np.argmin(losses100)]
print('Lowest Loss:',np.min(losses100))
comp=[]
for i in range(len(vecs)):
    comp.append(KBest.mu[int(KBest.labels[i])].reshape(5,5))
for i in range(0,reduced.shape[0],5):
    for j in range(0,reduced.shape[1],5):
        img[i:i+5,j:j+5]=comp.pop(0)
print('PSNR:',PSNR(np.array(night,dtype='float64'),img*255,data_range=255))
plt.imshow(img*255,cmap='Greys_r')
plt.axis('off')
plt.savefig("Images/Night100.png",bbox_inches='tight',dpi=300)
```

```
23:39:40
All Averages Found
```

```
Convergence Reached 39  Itterations  15.789890338564542
23:40:01
All Averages Found
Convergence Reached 49  Itterations  15.760609367131863
23:40:23
All Averages Found
Convergence Reached 63  Itterations  15.752109758558207
23:40:50
All Averages Found
Convergence Reached 40  Itterations  15.772294996070437
23:41:13
All Averages Found
Convergence Reached 46  Itterations  15.790031383068852
23:41:36
All Averages Found
Convergence Reached 49  Itterations  15.801533610313625
23:42:02
All Averages Found
Convergence Reached 71  Itterations  15.764548581238649
23:42:33
All Averages Found
Convergence Reached 62  Itterations  15.802257428457702
23:43:00
All Averages Found
Convergence Reached 44  Itterations  15.824390540232411
23:43:25
All Averages Found
Convergence Reached 54  Itterations  15.795914253899113
Lowest Loss: 15.752109758558207
PSNR: 19.25852671236476
```

[84]:
```
# same as prv cell b ut with C=1000
runs1000=[]
losses1000=[]
for i in range(10):
    print(datetime.now().strftime("%H:%M:%S"))
    K=KmeanVQ(vecs,'K++',C=1000)
    K.run(itt=1000)
    runs1000.append(K)
    losses1000.append(K.Loss(norm=True))
KBest=runs1000[np.argmin(losses1000)]
print('Lowest Loss:',np.min(losses1000))
comp=[]
for i in range(len(vecs)):
    comp.append(KBest.mu[int(KBest.labels[i])].reshape(5,5))
for i in range(0,reduced.shape[0],5):
    for j in range(0,reduced.shape[1],5):
        img[i:i+5,j:j+5]=comp.pop(0)
print('PSNR:',PSNR(np.array(night,dtype='float64'),img*255,data_range=255))
plt.imshow(img*255,cmap='Greys_r')
plt.axis('off')
plt.savefig("Images/Night1000.png",bbox_inches='tight',dpi=300)
```

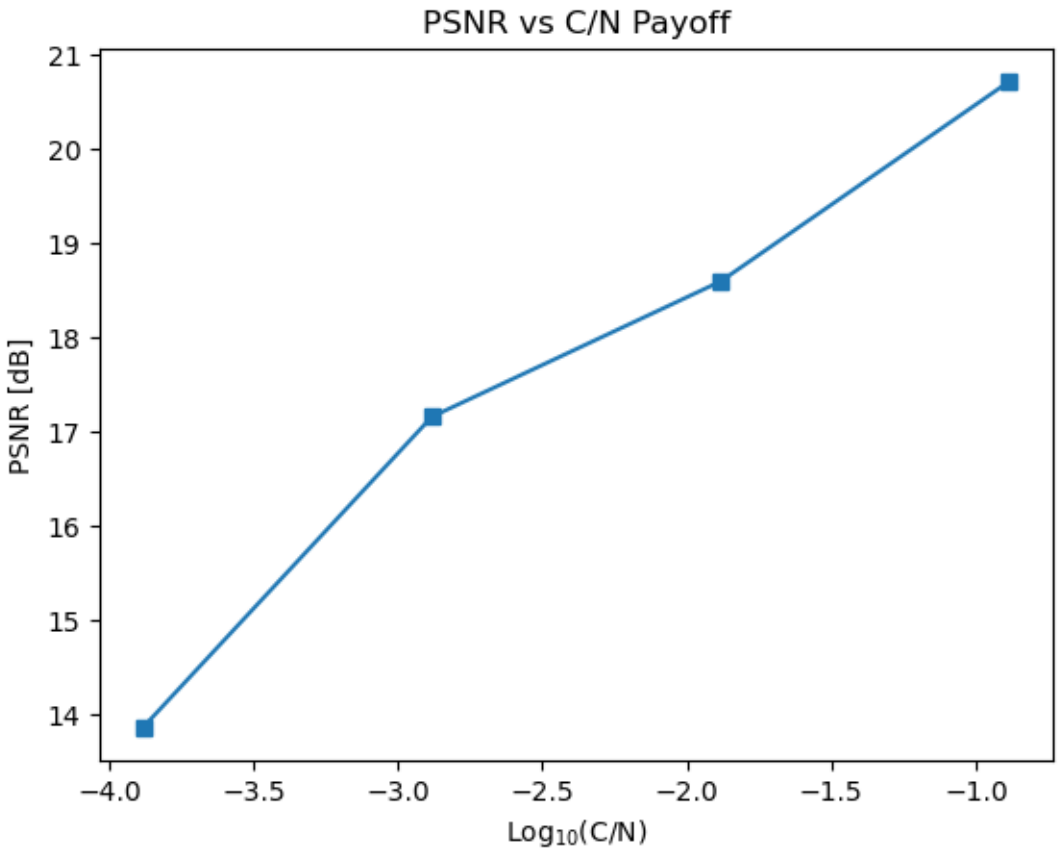
```
23:49:11
All Averages Found
```

```
Convergence Reached 21  Itterations  9.073355805118867
00:28:36
All Averages Found
Convergence Reached 25  Itterations  9.160524889992121
01:06:15
All Averages Found
Convergence Reached 14  Itterations  9.188742022239882
01:43:46
All Averages Found
Convergence Reached 16  Itterations  9.144661571651984
02:21:43
All Averages Found
Convergence Reached 20  Itterations  9.143589363711678
03:00:04
All Averages Found
Convergence Reached 24  Itterations  9.111121062463146
03:38:29
All Averages Found
Convergence Reached 15  Itterations  9.160862710555941
04:16:30
All Averages Found
Convergence Reached 19  Itterations  9.150172517481177
04:54:37
All Averages Found
Convergence Reached 17  Itterations  9.164266567055682
05:32:46
All Averages Found
Convergence Reached 24  Itterations  9.105575812927002
Lowest Loss: 9.073355805118867
PSNR: 21.6542345872515
```

[4]:
```
#plpotting orignal image so that is in the same shape and format as other ones
plt.imshow(night,cmap='Greys_r')
plt.axis('off')
plt.savefig("Images/NightNonCompress.png",bbox_inches='tight',dpi=300)
```

[16]:
```
CompPsnr=ascii.read('Task4Table.csv')#reading in table because I ddint wanna␣
↪rerun everything and i had forgot to save PSNR
```

[15]:
```
#plotting compreesion trade off plot
plt.plot(np.log10(CompPsnr['Comp. Quality']),CompPsnr['PSNR'],marker='s')
plt.xlabel(r'Log$_{10}$(C/N)')
plt.ylabel('PSNR [dB]')
plt.title('PSNR vs C/N Payoff')
plt.savefig('Images/PSNRPlt.png',dpi=300)
```


PSNR vs C/N Payoff
PSNR [dB]
21
20
19
18
17
16
15
14
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
Log10(C/N)

[ ]: