Sina Mahbobi

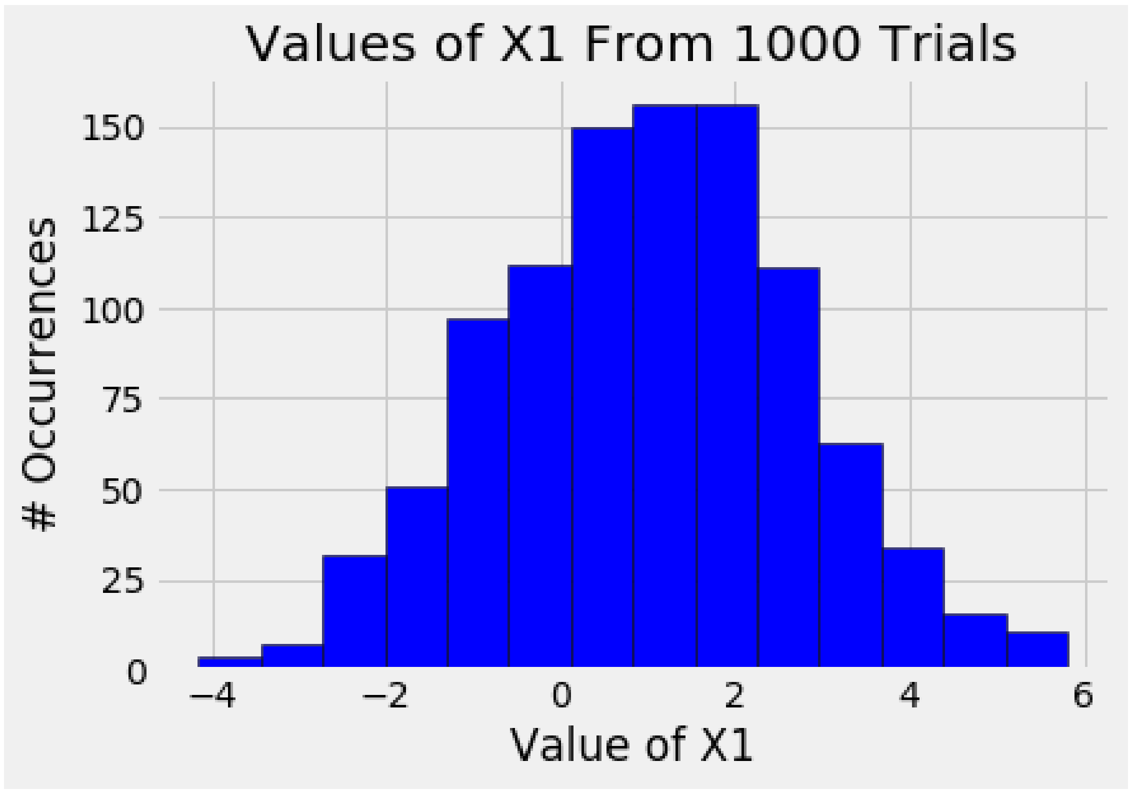
April 7, 2020

1 EE511 Project 7

1.0.1 Question 1

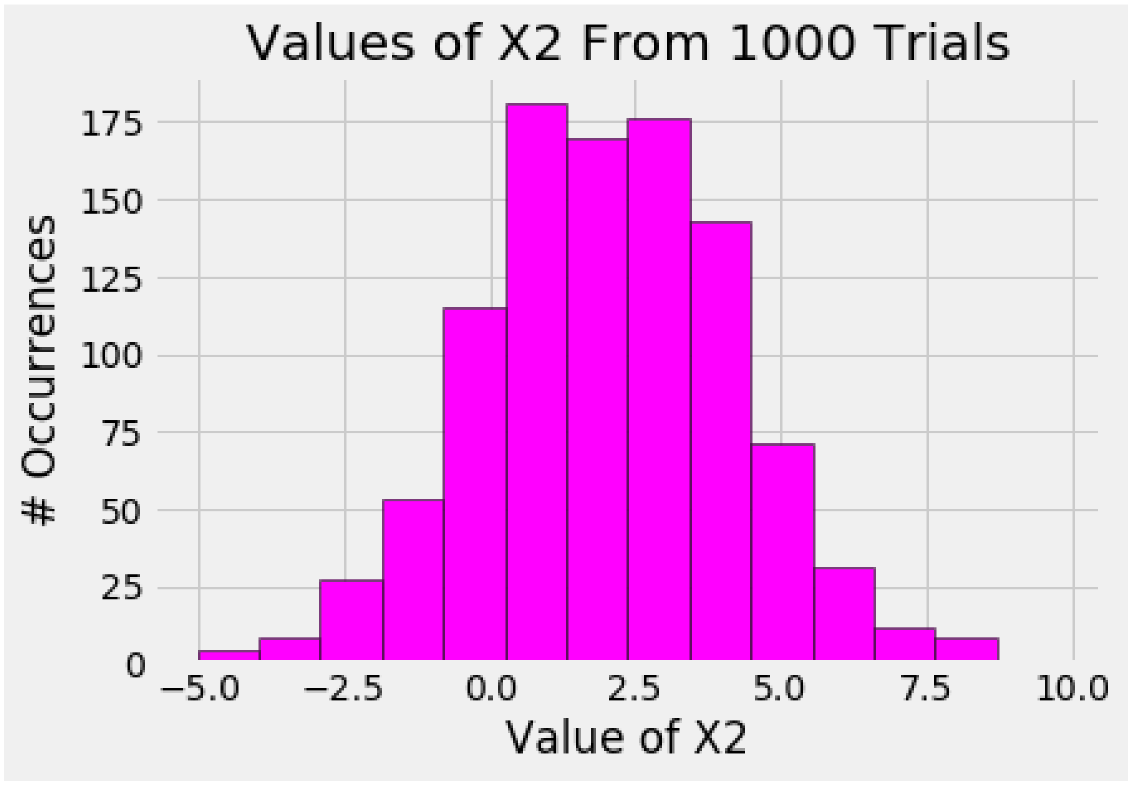
[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[93]:
```
sum_vals = np.empty(1000)
x1_vals = np.empty(1000)
x2_vals = np.empty(1000)
x3_vals = np.empty(1000)
for i in range(0,1000):
    mu = np.array([[1],[2],[3]])
    sigma = np.array([[3, -1, 1], [-1, 5, 3], [1, 3, 4]])
    z = np.array([[np.random.standard_normal()],  [np.random.standard_normal()],␣
 ↪ [np.random.standard_normal()]])
    A = np.linalg.cholesky(sigma)

    x = (np.dot(A,z) + mu).transpose()
    x = x[0]

    x1_vals[i] = x[0]
    x2_vals[i] = x[1]
    x3_vals[i] = x[2]

    sum_vals[i] = x[0] + x[1] + x[2]

plt.style.use('fivethirtyeight')

plt.hist(x1_vals, bins = 14, edgecolor = 'black', facecolor = 'blue' )
plt.xlabel("Value of X1")
plt.ylabel("# Occurrences")
plt.title("Values of X1 From 1000 Trials")
plt.show()
mean_x1 = x1_vals.mean()
print("The mean of X1 from 1000 trials: ", str(mean_x1))
var_x1 = x1_vals.var()
print("The variance of X1 from 1000 trials: ", str(var_x1))
```



```
The mean of X1 from 1000 trials:  1.0075443407403795
The variance of X1 from 1000 trials:  2.9831282347212524
```

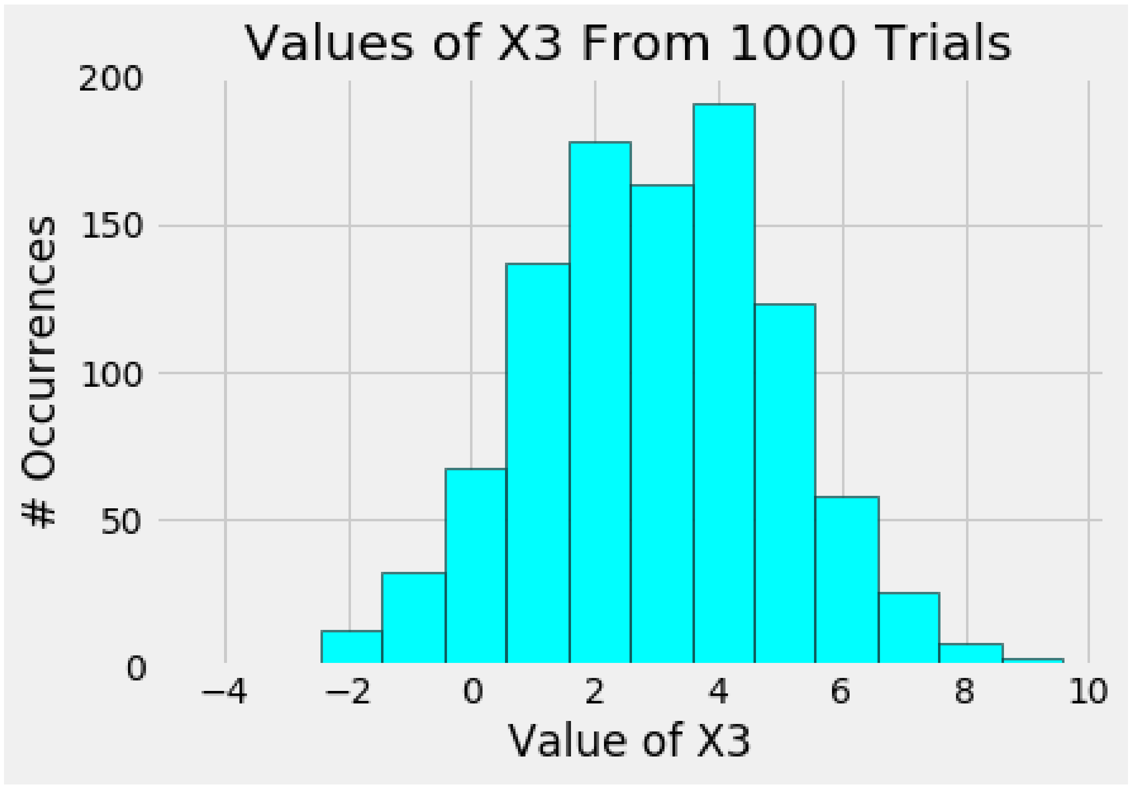
[94]:
```
plt.hist(x2_vals, bins = 14, edgecolor = 'black', facecolor = 'magenta' )
plt.xlabel("Value of X2")
plt.ylabel("# Occurrences")
plt.title("Values of X2 From 1000 Trials")
plt.show()
mean_x2 = x2_vals.mean()
print("The mean of X2 from 1000 trials: ", str(mean_x2))
var_x2 = x2_vals.var()
print("The variance of X2 from 1000 trials: ", str(var_x2))
```



```
The mean of X2 from 1000 trials:  2.028523479110944
The variance of X2 from 1000 trials:  4.847207017947193
```

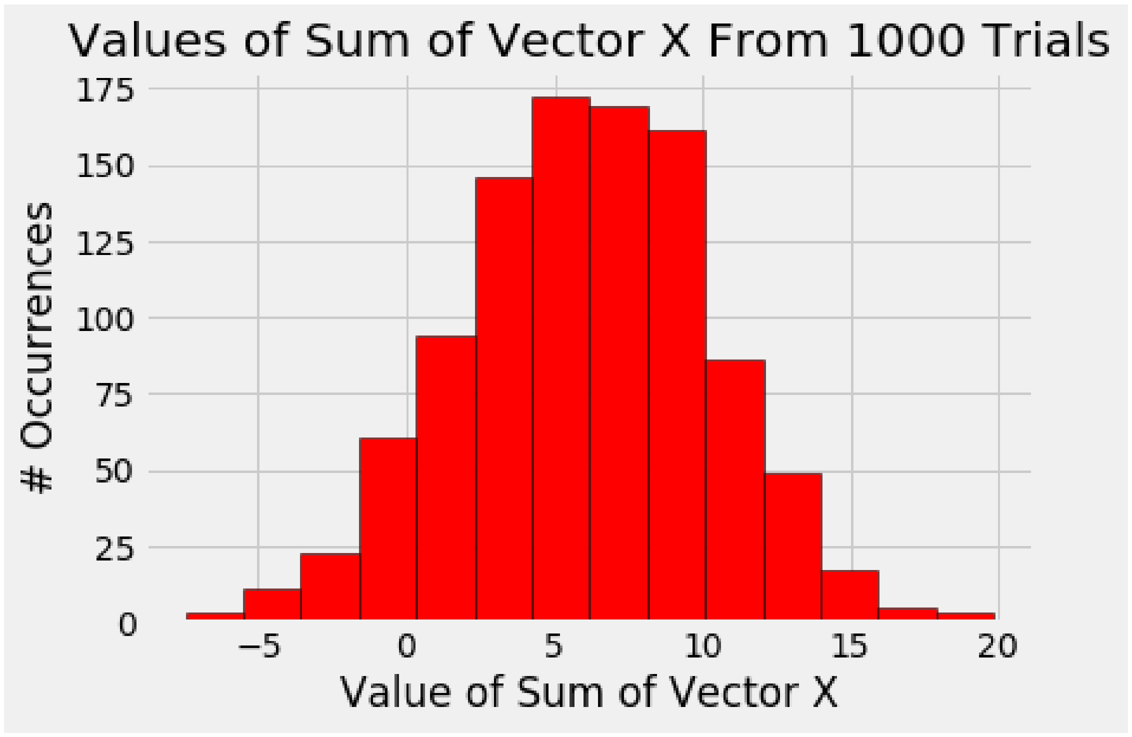
[95]:

```
plt.hist(x3_vals, bins = 14, edgecolor = 'black', facecolor = 'cyan' )
plt.xlabel("Value of X3")
plt.ylabel("# Occurrences")
plt.title("Values of X3 From 1000 Trials")
plt.show()
mean_x3 = x3_vals.mean()
print("The mean of X2 from 1000 trials: ", str(mean_x3))
var_x3 = x3_vals.var()
print("The variance of X3 from 1000 trials: ", str(var_x3))
```



```
The mean of X2 from 1000 trials:  2.9875537325921266
The variance of X3 from 1000 trials:  4.068478808142548
```

[96]:
```
plt.hist(sum_vals, bins = 14, edgecolor = 'black', facecolor = 'red' )
plt.xlabel("Value of Sum of Vector X")
plt.ylabel("# Occurrences")
plt.title("Values of Sum of Vector X From 1000 Trials")
plt.show()
mean_x = sum_vals.mean()
print("The mean of X Vector from 1000 trials: ", str(mean_x))
print("The sum of X1, X2, X3 mean: ", str(mean_x1 + mean_x2+ mean_x3))

var_sum = sum_vals.var()
print("The variance of Vector X from 1000 trials: ", str(var_sum))
print("The sum of X1, X2, X3 variance: ", str(var_x1 + var_x2+ var_x3))
```



```
The mean of X Vector from 1000 trials:  6.02362155244345
The sum of X1, X2, X3 mean:  6.02362155244345
The variance of Vector X from 1000 trials:  18.056389821416154
The sum of X1, X2, X3 variance:  11.898814060810995
```

1.0.2 Question 1 Analysis

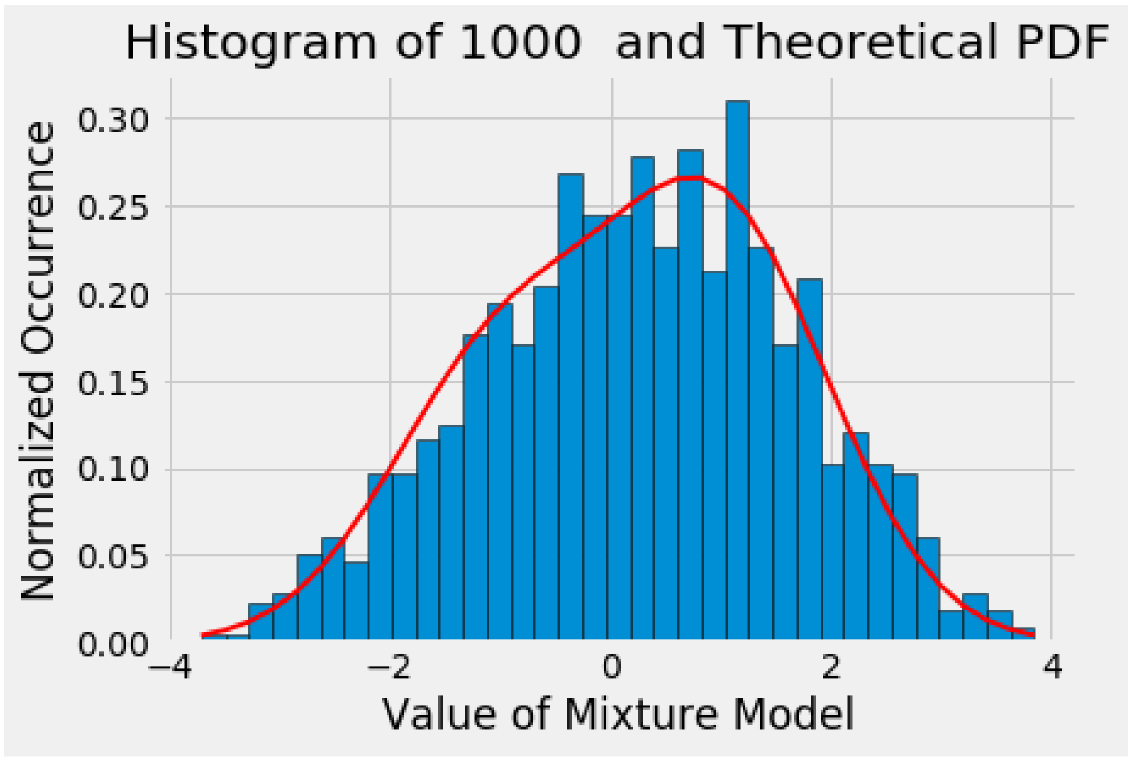
- The first histogram is that of the values of X1 generated from 1000 trials
- The second histogram is that of the values of X2 generated from 1000 trials
- The third histogram is that of the values of X3 generated from 1000 trials
- The fourth histogram is that of the values of X3 generated from 1000 trials
- The means of each individual Xi sum to the mean of the vector, which is correct because the sum of normal RVs is also a normal RV. This is not the case for the variances however, because the combinations of Xi have covariances, and are thus not independent RVs. If they were independent, the covariances would be zero and the three variances of the Xi's would sum to equal the variance of the vector.

```
[ ]:
```

1.0.3 Question 2

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[36]:
```
N = 1000
mu1, sigma1 = -1, 1 # mean and standard deviation of the two models
mu2, sigma2 = 1, 1
p = 0.4
s = np.empty(N)

for i in range(N):
    r = np.random.uniform()
    #print(r)
    if r < p:
        s[i] = np.random.normal(mu1, sigma1)
    else:
        s[i] = np.random.normal(mu2, sigma2)
    #print(s[i])
plt.style.use('fivethirtyeight')
count, bins, ignored = plt.hist(s, 35, density=True, edgecolor = 'black')


plt.plot(bins, p*(1/(sigma1 * np.sqrt(2 * np.pi)) *
                np.exp( - (bins - mu1)**2 / (2 * sigma1**2) )) + (1-p)*(1/
 ↪(sigma2 * np.sqrt(2 * np.pi)) *
                np.exp( - (bins - mu2)**2 / (2 * sigma2**2)) ),
         linewidth=2, color='r')

plt.xlabel("Value of Mixture Model")
plt.ylabel("Normalized Occurrence")
plt.title("Histogram of 1000  and Theoretical PDF")
plt.show()
```



1.0.4 Question 2 Analysis

- Above is a histogram of 1000 samples of the mixture distribution: f(x) = 0.4 * N(-1,1) + 0.6 * N(1,1) normalized so that the theoretical pdf can be overlaid. The red line is the theoretical pdf.
- In this case, it is hard to see the different bumps where the means are because the means of the two different Gaussian models are close to one another, with small variance. If the means were further apart these two humps would be more noticeable.

[ ]:

```

```

1.0.5 Question 3

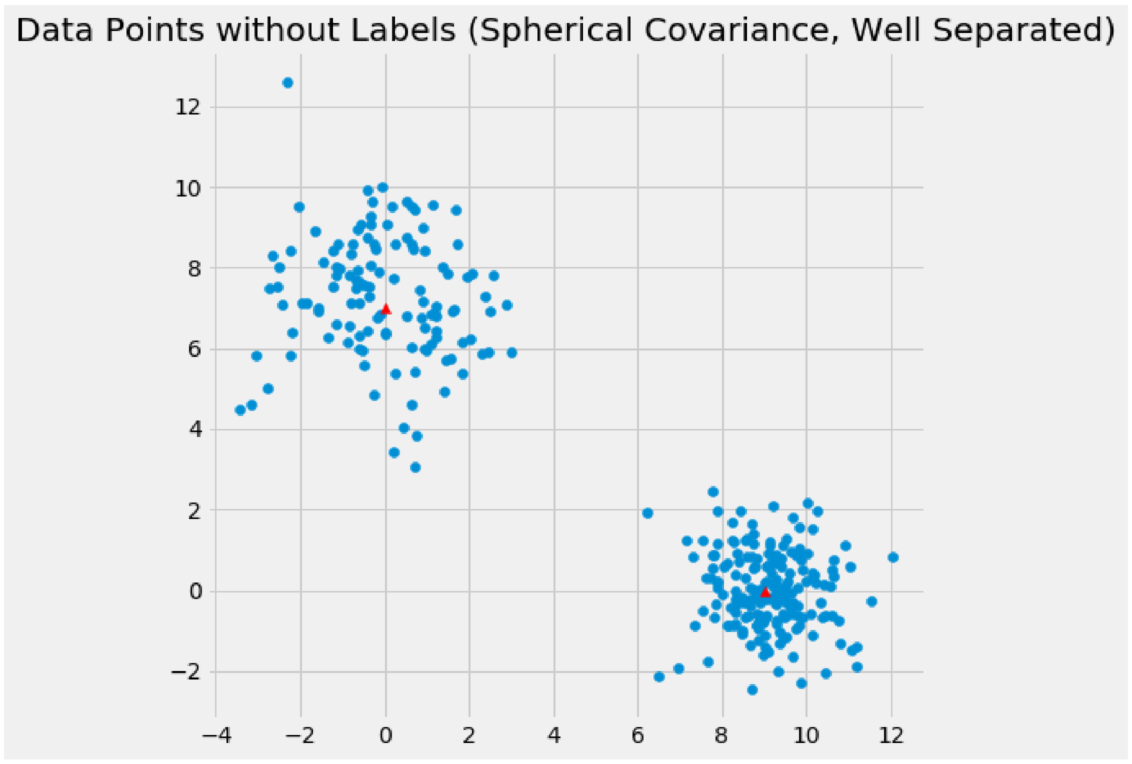
[1]:

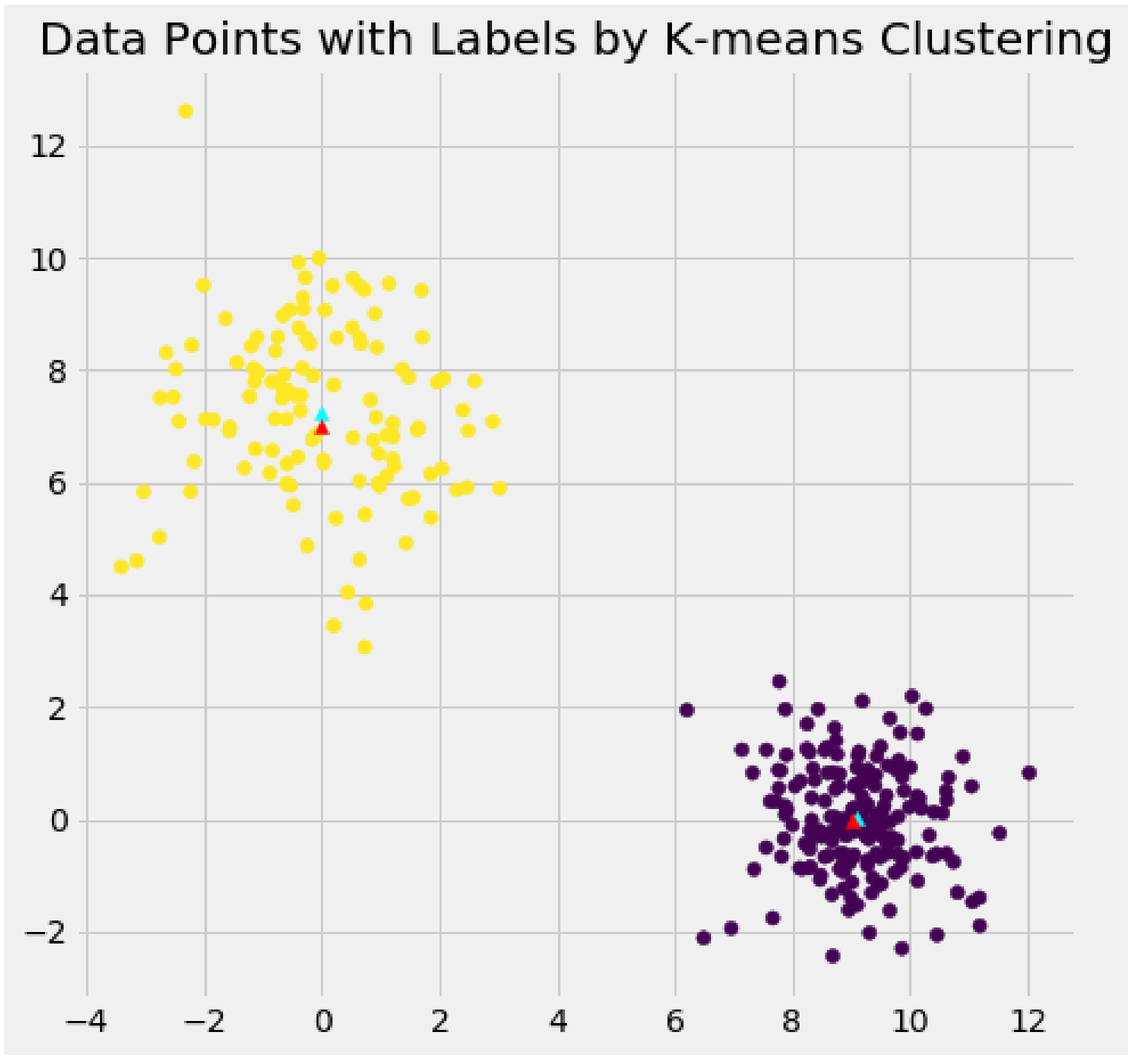
```python
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal
```

[2]:

```python
# generate data x with spherical covariance, well separated
N = 300
mu1 = np.array([9,0])
sigma1 = np.array([[1, 0],[0, 1]])
```

```
mu2 = np.array([0, 7])
sigma2 = np.array([[2, 0], [0, 2]])
weights = [0.6, 0.4]


np.random.seed(1) #For reproducibility
r1 = np.random.multivariate_normal(mu1, sigma1,int(weights[0]*N))
r2 = np.random.multivariate_normal(mu2, sigma2,int(weights[1]*N))

X = np.vstack((r1,r2)) #Cascade data points
np.random.shuffle(X)

plt.style.use('fivethirtyeight')
plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1])
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')
plt.title('Data Points without Labels (Spherical Covariance, Well Separated)')
plt.show()

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_.astype(float))
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],␣
 ↪marker='^', c='cyan')
# uncomment below to plot the mean values from each distribution
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')

plt.title('Data Points with Labels by K-means Clustering')
plt.show()
```


Data Points without Labels (Spherical Covariance, Well Separated)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4
6
8
10
12



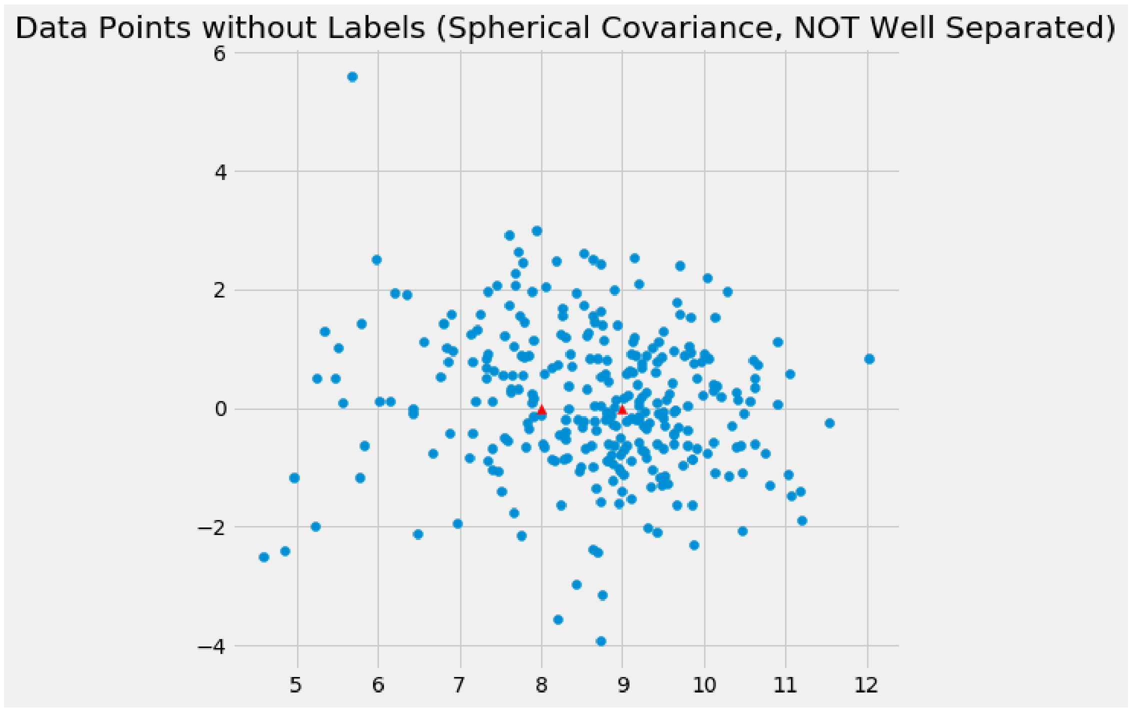
[3]:
```
# initialization

w = np.array([0.5, 0.5])
u1 = np.array(kmeans.cluster_centers_[0])
u2 = np.array(kmeans.cluster_centers_[1])
cov1 = np.eye(2)
cov2 = np.eye(2)
L_before = 0
gamma1 = np.empty(300)
gamma2 = np.empty(300)
gamma1_x_sum = np.empty(300)
gamma2_x_sum = np.empty(300)

for k in range(0,N):
    L_before += (np.log(w[0]*multivariate_normal.pdf(X[k], u1, cov1)) \
                +np.log(w[1]*multivariate_normal.pdf(X[k], u2, cov2) ) )

L_before = L_before / N
L_after = 0
iterations = 0
threshold = 1e-4

while np.abs(L_after - L_before) > threshold :

    iterations += 1
    L_before = L_after
    #for loop in range(0,5):
    # E Step
    for i in range(0,N):
        gamma1[i] = (w[0]*multivariate_normal.pdf(X[i], u1, cov1)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))


        gamma2[i] = (w[1]*multivariate_normal.pdf(X[i], u2, cov2)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))

    # Now, M Step
    u1_num = 0
    u2_num = 0
    for j in range(0,N):
        u1_num = u1_num + gamma1[j]*X[j]
        u2_num = u2_num + gamma2[j]*X[j]

    u1 = u1_num / np.sum(gamma1)
    u2 = u2_num / np.sum(gamma2)

    cov1_num = np.zeros([2,2])
    cov2_num = np.zeros([2,2])

    for k in range(0,N):
        cov1_num = cov1_num + gamma1[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u1, (-1,1)) )*(X[k] - u1)
        cov2_num = cov2_num + gamma2[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u2, (-1,1)) )*(X[k] - u2)

    cov1 = cov1_num / np.sum(gamma1)
    cov2 = cov2_num / np.sum(gamma2)

    w[0] = np.sum(gamma1)/N
    w[1] = np.sum(gamma2)/N
    L_after = 0
    for l in range(0,N):
        L_after = L_after + (np.log(w[0]*multivariate_normal.pdf(X[l], u1,␣
 ↪cov1)) \
            +np.log(w[1]*multivariate_normal.pdf(X[l], u2, cov2) ) )
    L_after = L_after / N
print("Well Separated, Spherical Covariance \n")
print("Number of Iterations: ", str(iterations))
print("Theoretical mu1: ", str(mu1))
print("Estimated mu1: ", str(u1))
print("\n")
print("Theoretical mu2: ", str(mu2))
print("Estimated mu2: ", str(u2))
print("\n")
print("Theoretical w1: ", str(weights[0]))
print("Estimated w1: ", str(w[0]))
print("\n")
print("Theoretical w2: ", str(weights[1]))
print("Estimated w2: ", str(w[1]))
print("Theoretical sigma1:\n", str(sigma1))
print("Estimated sigma1:\n", str(cov1))
print("\n")
print("Theoretical sigma2:\n", str(sigma2))
print("Estimated sigma2:\n", str(cov2))
```

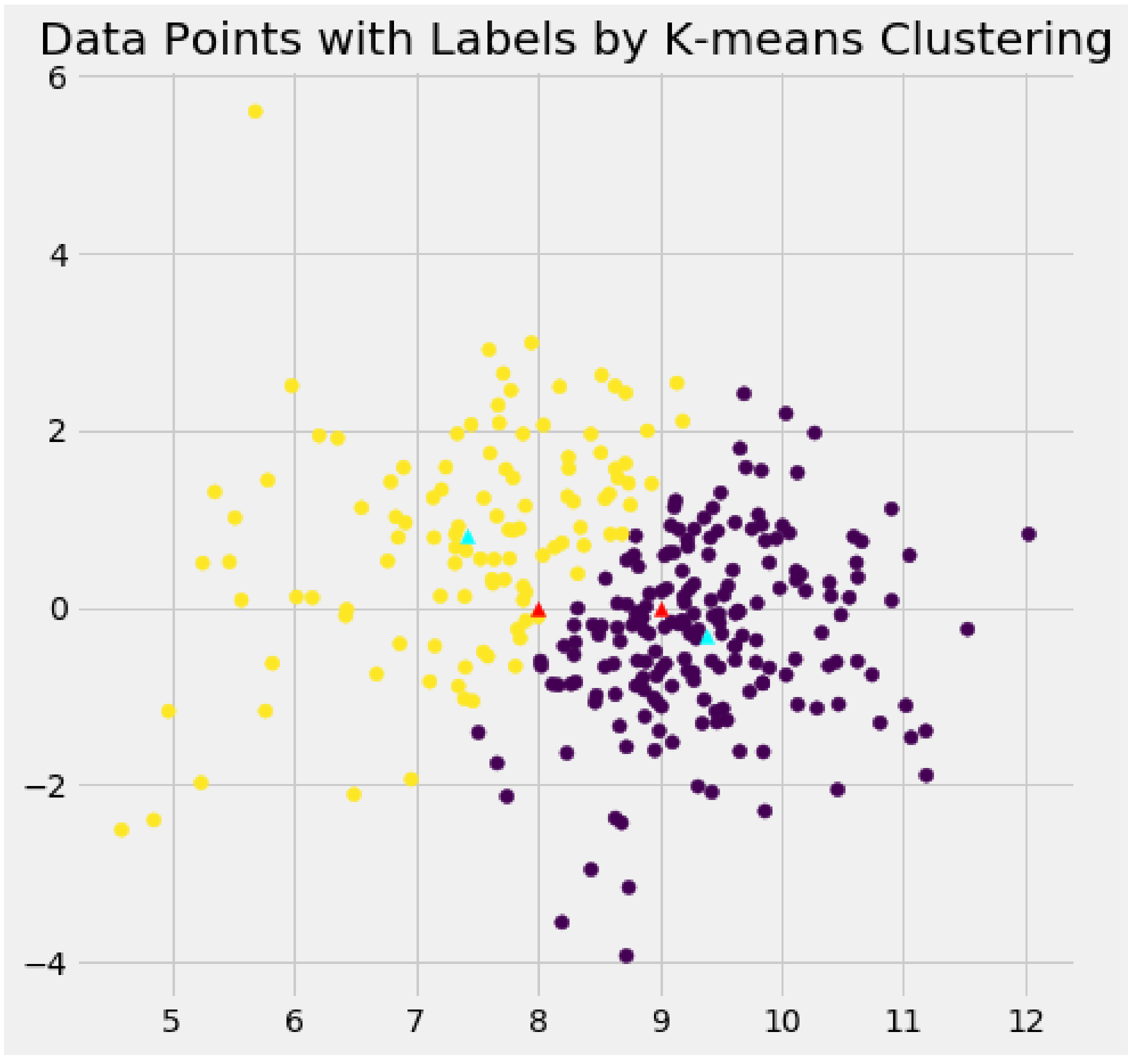
```
Well Separated, Spherical Covariance

Number of Iterations:  3
Theoretical mu1:  [9 0]
Estimated mu1:  [9.10866342e+00 7.11792127e-03]


Theoretical mu2:  [0 7]
Estimated mu2:  [-0.03274918  7.25003081]


Theoretical w1:  0.6
Estimated w1:  0.5999996618576436


Theoretical w2:  0.4
Estimated w2:  0.40000033814235647
Theoretical sigma1:
 [[1 0]
 [0 1]]
```

```
Estimated sigma1:
 [[ 0.89766837 -0.07758052]
 [-0.07758052  0.94333668]]


Theoretical sigma2:
 [[2 0]
 [0 2]]
Estimated sigma2:
 [[ 2.04035703 -0.23069911]
 [-0.23069911  2.34063416]]
```

[4]:

```python
# generate data x with spherical covariance, NOT well separated
N = 300
mu1 = np.array([9,0])
sigma1 = np.array([[1, 0],[0, 1]])
mu2 = np.array([8, 0])
sigma2 = np.array([[2, 0], [0, 2]])
weights = [0.6, 0.4]


np.random.seed(1) #For reproducibility
r1 = np.random.multivariate_normal(mu1, sigma1,int(weights[0]*N))
r2 = np.random.multivariate_normal(mu2, sigma2,int(weights[1]*N))

X = np.vstack((r1,r2)) #Cascade data points
np.random.shuffle(X)

plt.style.use('fivethirtyeight')
plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1])
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')
plt.title('Data Points without Labels (Spherical Covariance, NOT Well␣
↪Separated)')
plt.show()

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_.astype(float))
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],␣
↪marker='^', c='cyan')
# uncomment below to plot the mean values from each distribution
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')
plt.title('Data Points with Labels by K-means Clustering')
plt.show()
```



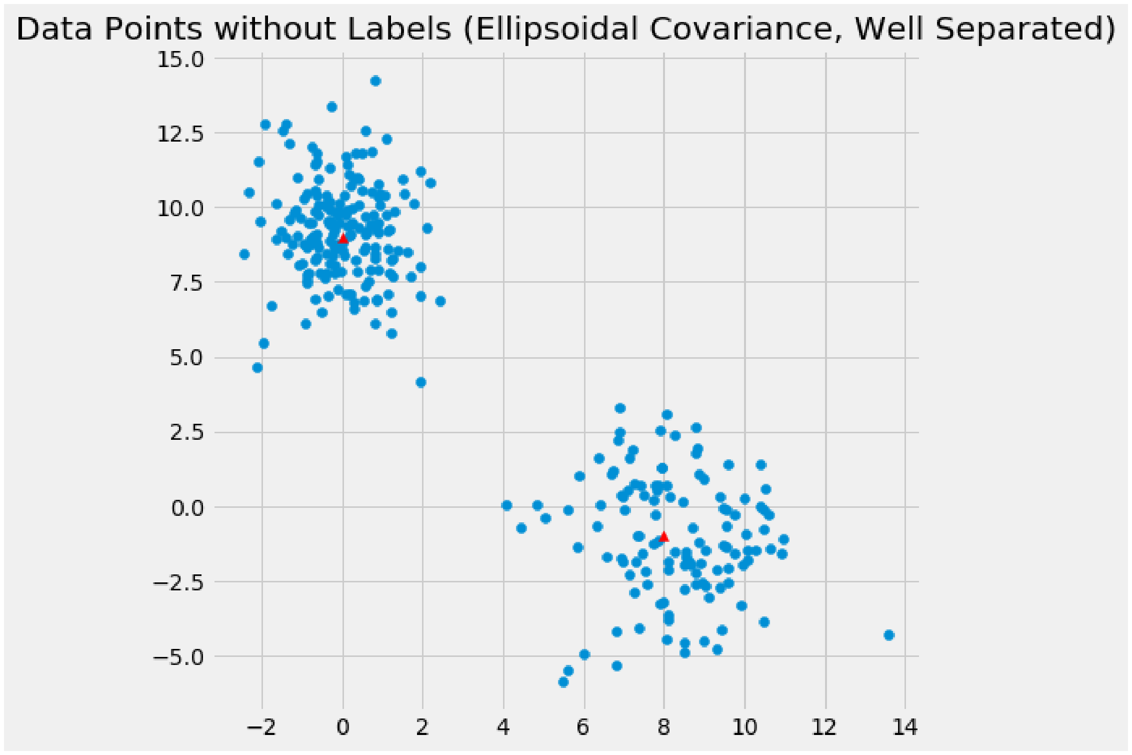


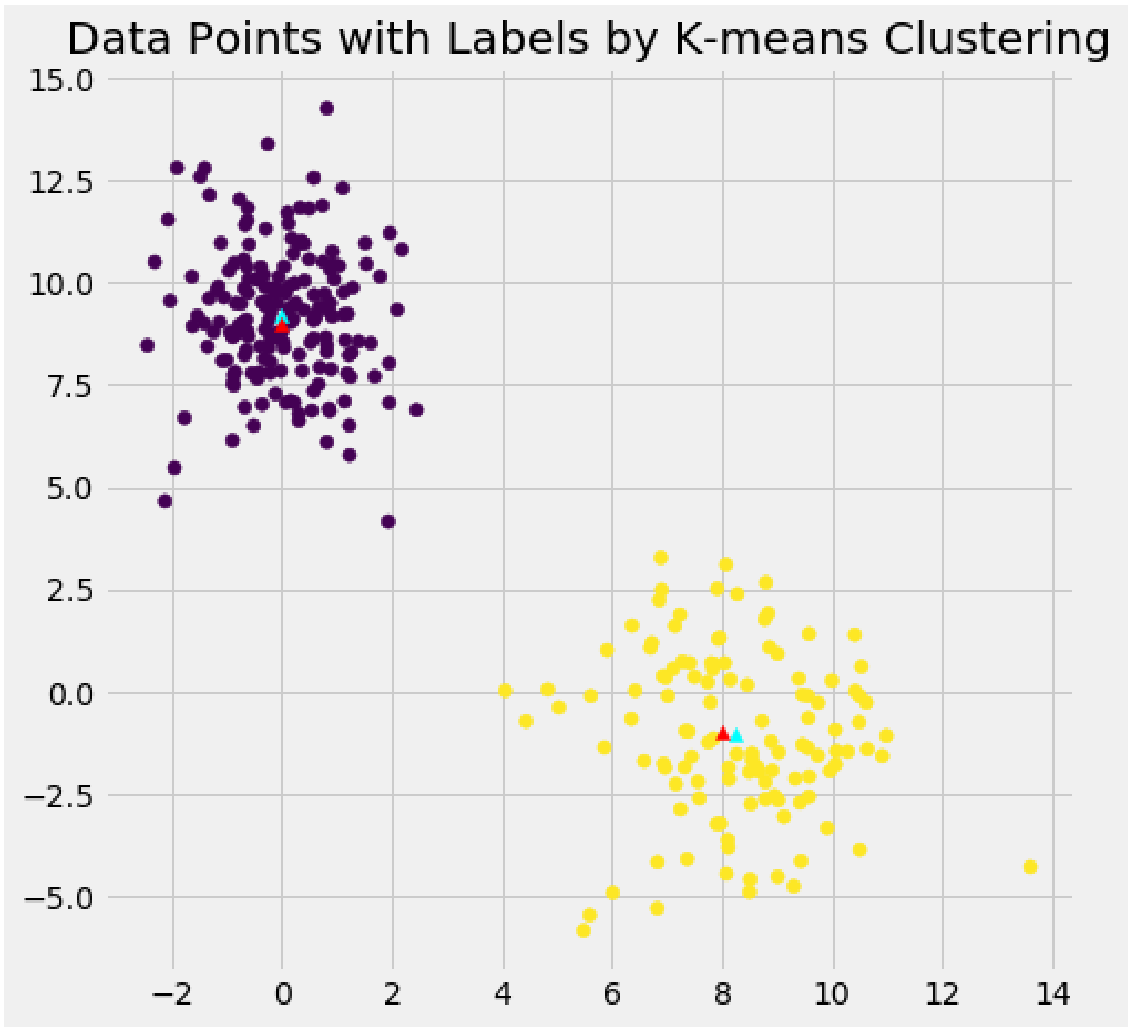
[5]:
```
# initialization

w = np.array([0.5, 0.5])
u1 = np.array(kmeans.cluster_centers_[0])
u2 = np.array(kmeans.cluster_centers_[1])
cov1 = np.eye(2)
cov2 = np.eye(2)
L_before = 0
gamma1 = np.empty(300)
gamma2 = np.empty(300)
gamma1_x_sum = np.empty(300)
gamma2_x_sum = np.empty(300)

for k in range(0,N):
    L_before += (np.log(w[0]*multivariate_normal.pdf(X[k], u1, cov1)) \
                +np.log(w[1]*multivariate_normal.pdf(X[k], u2, cov2) ) )

L_before = L_before / N
L_after = 0
iterations = 0
threshold = 1e-4

while np.abs(L_after - L_before) > threshold :

    iterations += 1
    L_before = L_after
    #for loop in range(0,5):
    # E Step
    for i in range(0,N):
        gamma1[i] = (w[0]*multivariate_normal.pdf(X[i], u1, cov1)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))


        gamma2[i] = (w[1]*multivariate_normal.pdf(X[i], u2, cov2)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))

    # Now, M Step
    u1_num = 0
    u2_num = 0
    for j in range(0,N):
        u1_num = u1_num + gamma1[j]*X[j]
        u2_num = u2_num + gamma2[j]*X[j]

    u1 = u1_num / np.sum(gamma1)
    u2 = u2_num / np.sum(gamma2)

    cov1_num = np.zeros([2,2])
    cov2_num = np.zeros([2,2])

    for k in range(0,N):
        cov1_num = cov1_num + gamma1[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u1, (-1,1)) )*(X[k] - u1)
        cov2_num = cov2_num + gamma2[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u2, (-1,1)) )*(X[k] - u2)

    cov1 = cov1_num / np.sum(gamma1)
    cov2 = cov2_num / np.sum(gamma2)

    w[0] = np.sum(gamma1)/N
    w[1] = np.sum(gamma2)/N
    L_after = 0
    for l in range(0,N):
        L_after = L_after + (np.log(w[0]*multivariate_normal.pdf(X[l], u1,␣
↪cov1)) \
            +np.log(w[1]*multivariate_normal.pdf(X[l], u2, cov2) ) )
    L_after = L_after / N
print("NOT Well Separated, Spherical Covariance \n")
print("Number of Iterations: ", str(iterations))
print("Theoretical mu1: ", str(mu1))
print("Estimated mu1: ", str(u1))
print("\n")
print("Theoretical mu2: ", str(mu2))
print("Estimated mu2: ", str(u2))
print("\n")
print("Theoretical w1: ", str(weights[0]))
print("Estimated w1: ", str(w[0]))
print("\n")
print("Theoretical w2: ", str(weights[1]))
print("Estimated w2: ", str(w[1]))
print("Theoretical sigma1:\n", str(sigma1))
print("Estimated sigma1:\n", str(cov1))
print("\n")
print("Theoretical sigma2:\n", str(sigma2))
print("Estimated sigma2:\n", str(cov2))
```

```
NOT Well Separated, Spherical Covariance

Number of Iterations:  215
Theoretical mu1:  [9 0]
Estimated mu1:  [8.97012115 0.1410203 ]


Theoretical mu2:  [8 0]
Estimated mu2:  [ 7.11183783 -0.07362771]


Theoretical w1:  0.6
Estimated w1:  0.8288604005522126


Theoretical w2:  0.4
Estimated w2:  0.17113959944778748
Theoretical sigma1:
 [[1 0]
 [0 1]]
```

```
Estimated sigma1:
 [[ 1.05215593 -0.25584767]
 [-0.25584767  1.15537897]]


Theoretical sigma2:
 [[2 0]
 [0 2]]
Estimated sigma2:
 [[ 1.78498279 -0.29147058]
 [-0.29147058  3.22675984]]
```

[6]:
```
# generate data x with Ellipsoidal covariance, well separated
N = 300
mu1 = np.array([0,9])
sigma1 = np.array([[1, 0],[0, 3]])
mu2 = np.array([8, -1])
sigma2 = np.array([[2, 0], [0, 4]])
weights = [0.6, 0.4]


np.random.seed(1) #For reproducibility
r1 = np.random.multivariate_normal(mu1, sigma1,int(weights[0]*N))
r2 = np.random.multivariate_normal(mu2, sigma2,int(weights[1]*N))

X = np.vstack((r1,r2)) #Cascade data points
np.random.shuffle(X)

plt.style.use('fivethirtyeight')
plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1])
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')
plt.title('Data Points without Labels (Ellipsoidal Covariance, Well Separated)')
plt.show()

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_.astype(float))
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],␣
 ↪marker='^', c='cyan')
# uncomment below to plot the mean values from each distribution
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')

plt.title('Data Points with Labels by K-means Clustering')
plt.show()
```


Data Points without Labels (Ellipsoidal Covariance, Well Separated)
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−2
0
2
4
6
8
10
12
14



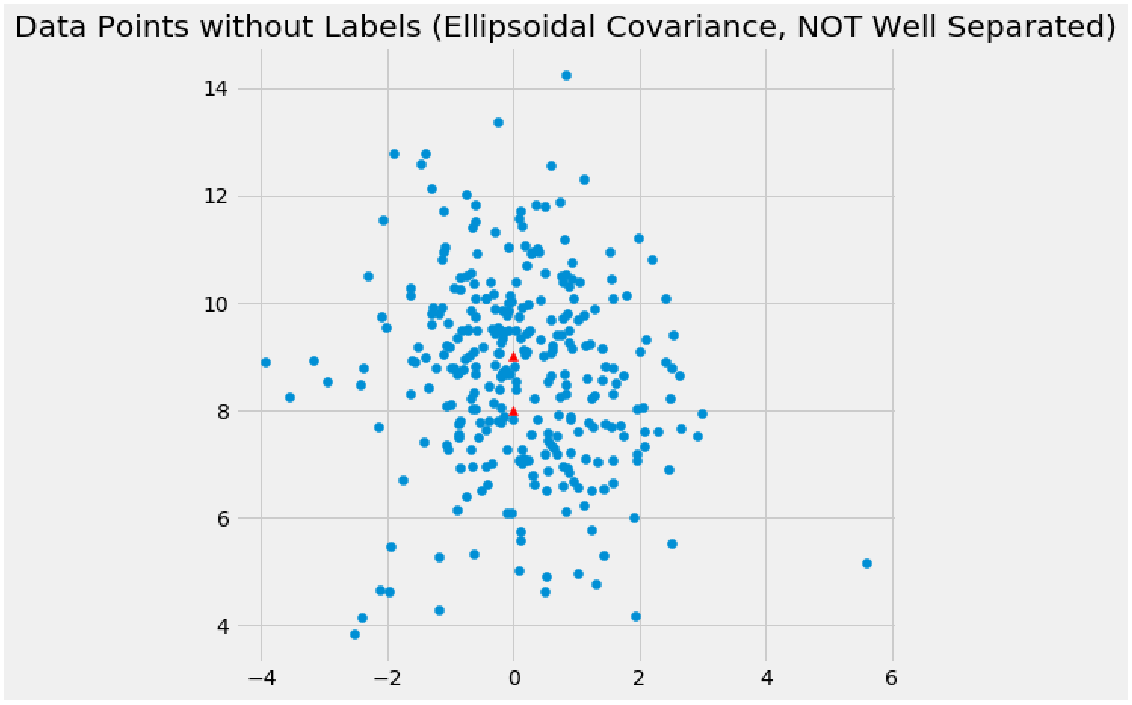
[7]:

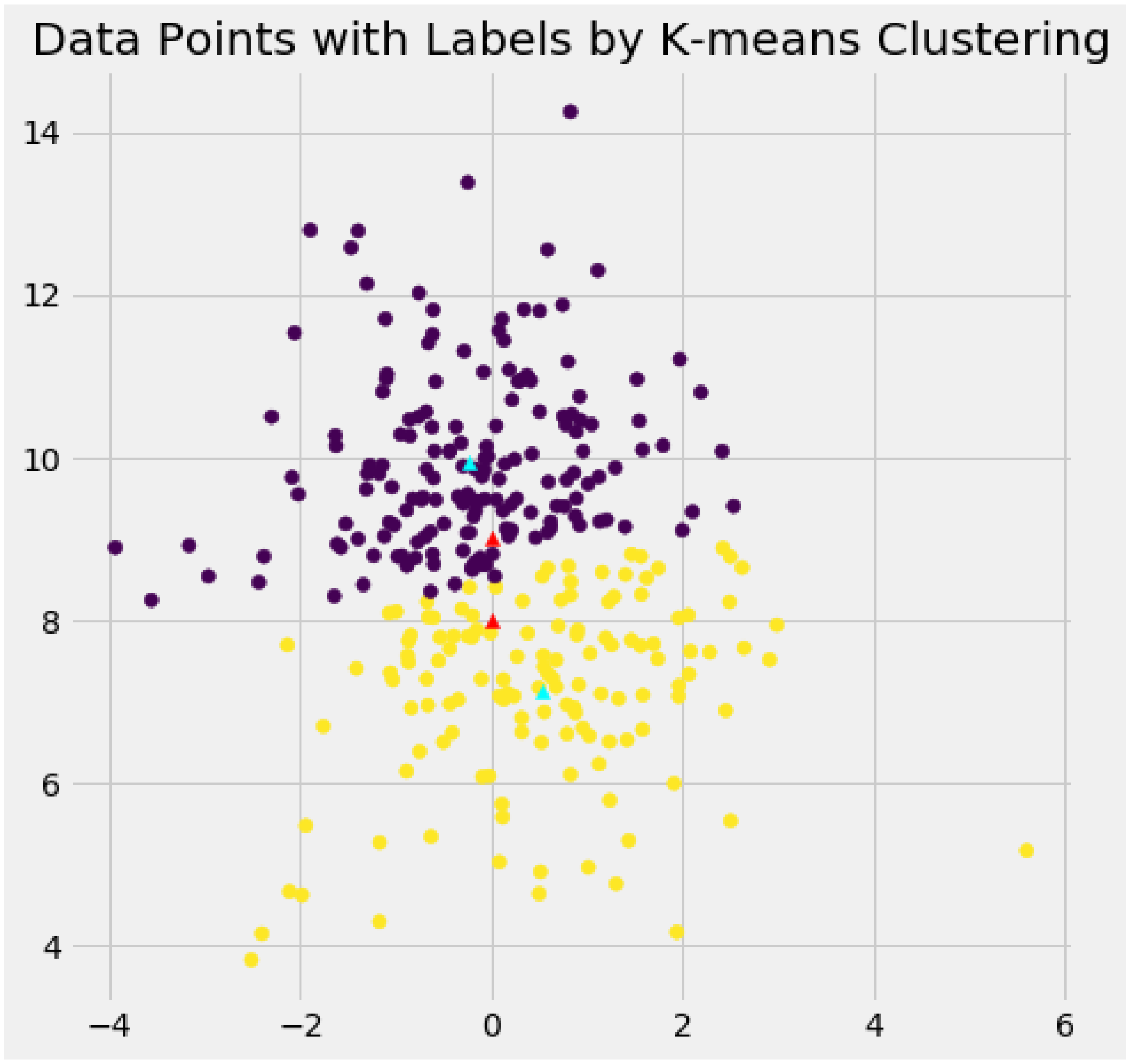
```
# initialization

w = np.array([0.5, 0.5])
u1 = np.array(kmeans.cluster_centers_[0])
u2 = np.array(kmeans.cluster_centers_[1])
cov1 = np.eye(2)
cov2 = np.eye(2)
L_before = 0
gamma1 = np.empty(300)
gamma2 = np.empty(300)
gamma1_x_sum = np.empty(300)
gamma2_x_sum = np.empty(300)

for k in range(0,N):
    L_before += (np.log(w[0]*multivariate_normal.pdf(X[k], u1, cov1)) \
                +np.log(w[1]*multivariate_normal.pdf(X[k], u2, cov2) ) )
```

```
L_before = L_before / N
L_after = 0
iterations = 0
threshold = 1e-4

while np.abs(L_after - L_before) > threshold :

    iterations += 1
    L_before = L_after
    #for loop in range(0,5):
    # E Step
    for i in range(0,N):
        gamma1[i] = (w[0]*multivariate_normal.pdf(X[i], u1, cov1)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))


        gamma2[i] = (w[1]*multivariate_normal.pdf(X[i], u2, cov2)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))

    # Now, M Step
    u1_num = 0
    u2_num = 0
    for j in range(0,N):
        u1_num = u1_num + gamma1[j]*X[j]
        u2_num = u2_num + gamma2[j]*X[j]

    u1 = u1_num / np.sum(gamma1)
    u2 = u2_num / np.sum(gamma2)

    cov1_num = np.zeros([2,2])
    cov2_num = np.zeros([2,2])

    for k in range(0,N):
        cov1_num = cov1_num + gamma1[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u1, (-1,1)) )*(X[k] - u1)
        cov2_num = cov2_num + gamma2[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u2, (-1,1)) )*(X[k] - u2)

    cov1 = cov1_num / np.sum(gamma1)
    cov2 = cov2_num / np.sum(gamma2)

    w[0] = np.sum(gamma1)/N
    w[1] = np.sum(gamma2)/N
    L_after = 0
    for l in range(0,N):
        L_after = L_after + (np.log(w[0]*multivariate_normal.pdf(X[l], u1,␣
 ↪cov1)) \
            +np.log(w[1]*multivariate_normal.pdf(X[l], u2, cov2) ) )
    L_after = L_after / N
print("Well Separated, Ellipsoidal Covariance \n")
print("Number of Iterations: ", str(iterations))
print("Theoretical mu1: ", str(mu1))
print("Estimated mu1: ", str(u1))
print("\n")
print("Theoretical mu2: ", str(mu2))
print("Estimated mu2: ", str(u2))
print("\n")
print("Theoretical w1: ", str(weights[0]))
print("Estimated w1: ", str(w[0]))
print("\n")
print("Theoretical w2: ", str(weights[1]))
print("Estimated w2: ", str(w[1]))
print("Theoretical sigma1:\n", str(sigma1))
print("Estimated sigma1:\n", str(cov1))
print("\n")
print("Theoretical sigma2:\n", str(sigma2))
print("Estimated sigma2:\n", str(cov2))
```

```
Well Separated, Ellipsoidal Covariance

Number of Iterations:  3
Theoretical mu1:  [0 9]
Estimated mu1:  [7.08273264e-03 9.18830163e+00]


Theoretical mu2:  [ 8 -1]
Estimated mu2:  [ 8.24985772 -1.04617458]


Theoretical w1:  0.6
Estimated w1:  0.5999887417579609


Theoretical w2:  0.4
Estimated w2:  0.40001125824203915
Theoretical sigma1:
 [[1 0]
 [0 3]]
Estimated sigma1:
 [[ 0.94328577 -0.13419994]
 [-0.13419994  2.69259741]]


Theoretical sigma2:
 [[2 0]
 [0 4]]
Estimated sigma2:
 [[ 2.34166692 -0.32713789]
 [-0.32713789  4.08130846]]
```

[8]:
```
# generate data x with Ellipsoidal covariance, NOT well separated
N = 300
mu1 = np.array([0,9])
sigma1 = np.array([[1, 0],[0, 3]])
mu2 = np.array([0, 8])
sigma2 = np.array([[2, 0], [0, 3]])
weights = [0.6, 0.4]


np.random.seed(1) #For reproducibility
r1 = np.random.multivariate_normal(mu1, sigma1,int(weights[0]*N))
r2 = np.random.multivariate_normal(mu2, sigma2,int(weights[1]*N))

X = np.vstack((r1,r2)) #Cascade data points
np.random.shuffle(X)

plt.style.use('fivethirtyeight')
plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1])
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')
plt.title('Data Points without Labels (Ellipsoidal Covariance, NOT Well␣
 ↪Separated)')
plt.show()

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

plt.figure(figsize=(7, 7))
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_.astype(float))
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],␣
 ↪marker='^', c='cyan')
# uncomment below to plot the mean values from each distribution
plt.scatter(mu1[0],mu1[1], marker='^', c='red')
plt.scatter(mu2[0],mu2[1], marker='^', c='red')

plt.title('Data Points with Labels by K-means Clustering')
plt.show()
```


Data Points without Labels (Ellipsoidal Covariance, NOT Well Separated)
14
12
10
8
6
4
−4
−2
0
2
4
6



[9]:
```
# initialization

w = np.array([0.5, 0.5])
u1 = np.array(kmeans.cluster_centers_[0])
u2 = np.array(kmeans.cluster_centers_[1])
cov1 = np.eye(2)
cov2 = np.eye(2)
L_before = 0
gamma1 = np.empty(300)
gamma2 = np.empty(300)
gamma1_x_sum = np.empty(300)
gamma2_x_sum = np.empty(300)

for k in range(0,N):
    L_before += (np.log(w[0]*multivariate_normal.pdf(X[k], u1, cov1)) \
                +np.log(w[1]*multivariate_normal.pdf(X[k], u2, cov2) ) )

L_before = L_before / N
L_after = 0
iterations = 0
threshold = 1e-4

while np.abs(L_after - L_before) > threshold :

    iterations += 1
    L_before = L_after
    #for loop in range(0,5):
    # E Step
    for i in range(0,N):
        gamma1[i] = (w[0]*multivariate_normal.pdf(X[i], u1, cov1)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))


        gamma2[i] = (w[1]*multivariate_normal.pdf(X[i], u2, cov2)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))

    # Now, M Step
    u1_num = 0
    u2_num = 0
    for j in range(0,N):
        u1_num = u1_num + gamma1[j]*X[j]
        u2_num = u2_num + gamma2[j]*X[j]

    u1 = u1_num / np.sum(gamma1)
    u2 = u2_num / np.sum(gamma2)

    cov1_num = np.zeros([2,2])
    cov2_num = np.zeros([2,2])

    for k in range(0,N):
        cov1_num = cov1_num + gamma1[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u1, (-1,1)) )*(X[k] - u1)
        cov2_num = cov2_num + gamma2[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u2, (-1,1)) )*(X[k] - u2)

    cov1 = cov1_num / np.sum(gamma1)
    cov2 = cov2_num / np.sum(gamma2)

    w[0] = np.sum(gamma1)/N
    w[1] = np.sum(gamma2)/N
    L_after = 0
    for l in range(0,N):
        L_after = L_after + (np.log(w[0]*multivariate_normal.pdf(X[l], u1,␣
 ↪cov1)) \
            +np.log(w[1]*multivariate_normal.pdf(X[l], u2, cov2) ) )
    L_after = L_after / N
print("NOT Well Separated, Ellipsoidal Covariance \n")
print("Number of Iterations: ", str(iterations))
print("Theoretical mu1: ", str(mu1))
print("Estimated mu1: ", str(u1))
print("\n")
print("Theoretical mu2: ", str(mu2))
print("Estimated mu2: ", str(u2))
print("\n")
print("Theoretical w1: ", str(weights[0]))
print("Estimated w1: ", str(w[0]))
print("\n")
print("Theoretical w2: ", str(weights[1]))
print("Estimated w2: ", str(w[1]))
print("Theoretical sigma1:\n", str(sigma1))
print("Estimated sigma1:\n", str(cov1))
print("\n")
print("Theoretical sigma2:\n", str(sigma2))
print("Estimated sigma2:\n", str(cov2))
```

```
NOT Well Separated, Ellipsoidal Covariance

Number of Iterations:  219
Theoretical mu1:  [0 9]
Estimated mu1:  [0.15675725 8.92628756]


Theoretical mu2:  [0 8]
Estimated mu2:  [-0.32693826  6.81154404]


Theoretical w1:  0.6
Estimated w1:  0.8915191086329253


Theoretical w2:  0.4
Estimated w2:  0.1084808913670746
Theoretical sigma1:
 [[1 0]
 [0 3]]
```

```
Estimated sigma1:
 [[ 1.17838537 -0.37383135]
 [-0.37383135  2.71127256]]


Theoretical sigma2:
 [[2 0]
 [0 3]]
Estimated sigma2:
 [[ 4.0857862  -0.28477588]
 [-0.28477588  3.24894691]]
```

1.0.6 Question 3 Analysis

- For this question I tested 4 different scenarios
- The first scenario is where the two components of the mixture model have spherical variance, and are very well separated.
  - In this case, it takes three iterations for the difference between the log likelihood to go below the threshold, and the mean, weights, and covariance predictions are fairly accurate.
- The second scenario also has spherical covariance, but the data is not well separated.
  - In this case, it takes 215 iterations to go below the threshold. The weights are not predicted well, and the means and covariances aren't as accurate either.
- The third scenario is where the two components of the mixture model have ellipsoidal variance, and are very well separated.
  - In this case, it takes three iterations for the difference between the log likelihood to go below the threshold, and the mean, weights, and covariance predictions are fairly accurate.
- The fourth scenario also has ellipsoidal covariance, but the data is not well separated.
  - In this case, it takes 219 iterations to go below the threshold. The weights are not predicted well, and the means and covariances aren't as accurate either.

[ ]:

```

```

1.0.7 Question 4

[1]:

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
```

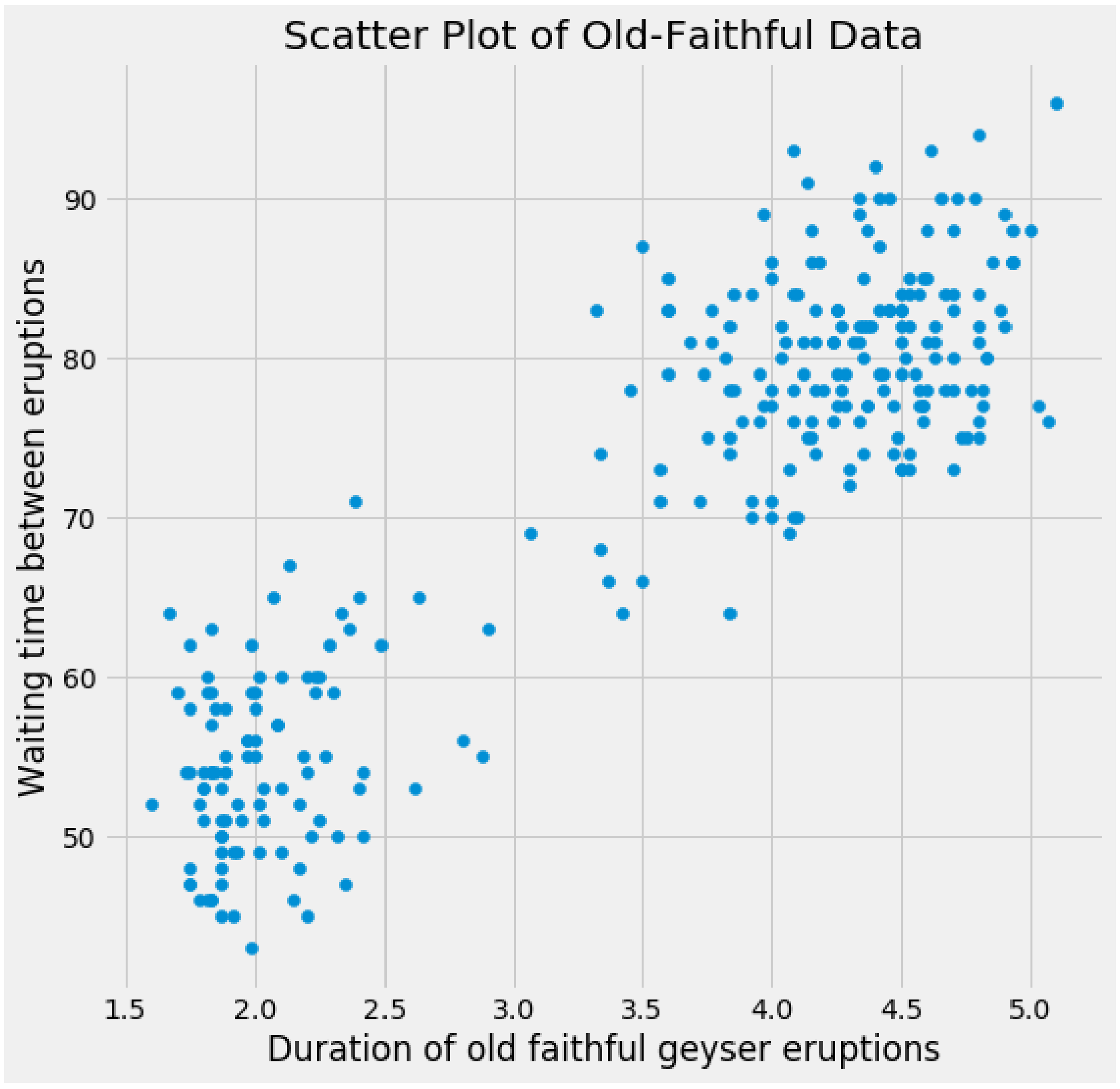
[2]:

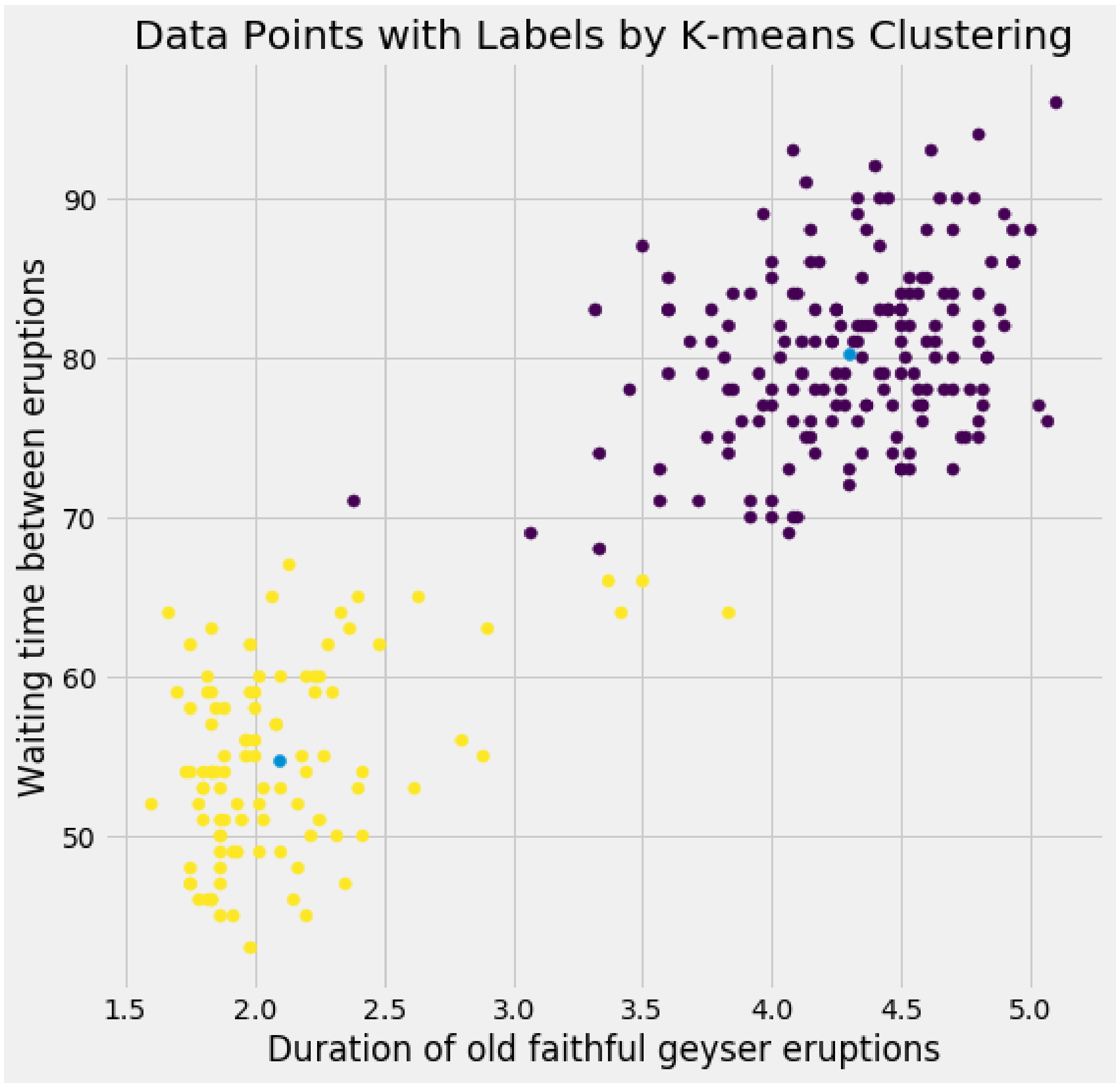
```python
df = pd.read_table('faithful.dat.txt', delim_whitespace=True,  \
    names=('Eruptions', 'Waiting'), skiprows = 26)
```

```
df
```

```
[2]:      Eruptions  Waiting
     1        3.600       79
     2        1.800       54
     3        3.333       74
     4        2.283       62
     5        4.533       85
     ..         ...      ...
     268      4.117       81
     269      2.150       46
     270      4.417       90
     271      1.817       46
     272      4.467       74

     [272 rows x 2 columns]
```

```
[3]: plt.style.use('fivethirtyeight')
     plt.figure(figsize=(8, 8))
     eruptions = df.iloc[:,0].values
     waiting = df.iloc[:,1].values
     plt.scatter(eruptions, waiting)
     plt.title("Scatter Plot of Old-Faithful Data")
     plt.xlabel("Duration of old faithful geyser eruptions")
     plt.ylabel("Waiting time between eruptions")
     plt.show()

     X = df.iloc[:,[0,1]].values #Cascade data points
     np.random.shuffle(X)
     kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
     plt.figure(figsize=(8, 8))
     plt.scatter(X[:,0], X[:,1], c=kmeans.labels_.astype(float))
     plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1])
     plt.xlabel("Duration of old faithful geyser eruptions")
     plt.ylabel("Waiting time between eruptions")
     plt.title('Data Points with Labels by K-means Clustering')
     plt.show()
```


Scatter Plot of Old-Faithful Data
Waiting time between eruptions
Duration of old faithful geyser eruptions
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

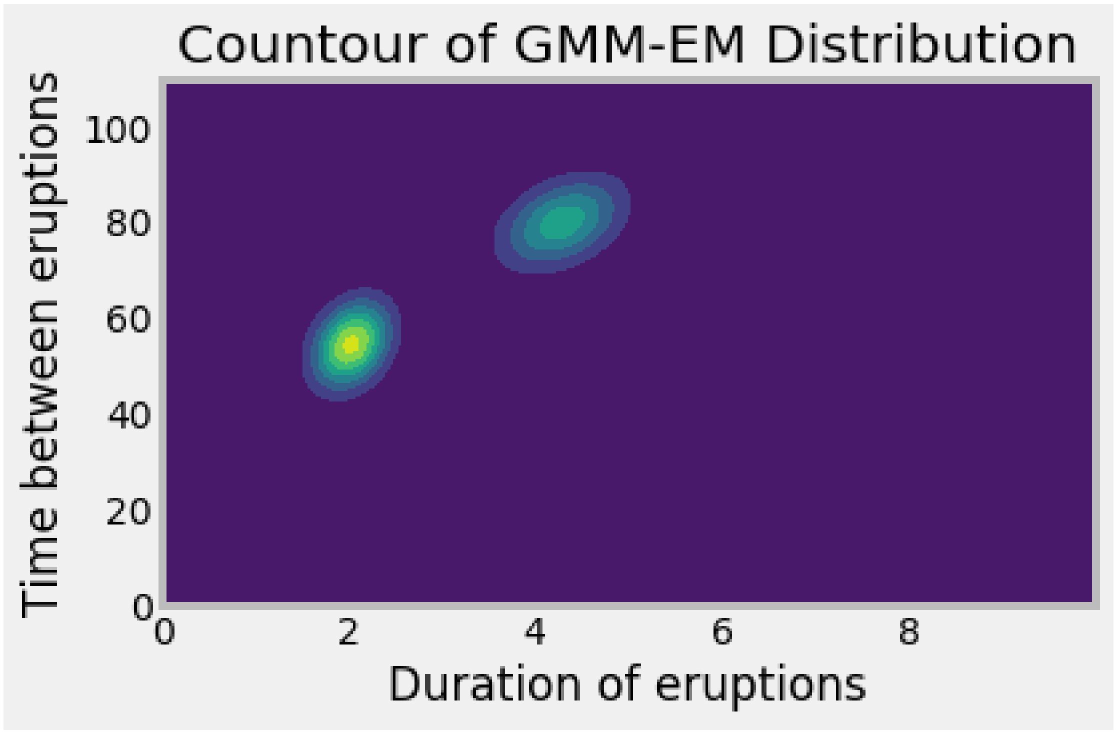


[4]:
```
# initialization
N = 272
w = np.array([0.5, 0.5])
u1 = np.array(kmeans.cluster_centers_[0])
u2 = np.array(kmeans.cluster_centers_[1])
cov1 = np.eye(2)
cov2 = np.eye(2)
L_before = 0
gamma1 = np.empty(N)
gamma2 = np.empty(N)
gamma1_x_sum = np.empty(N)
gamma2_x_sum = np.empty(N)

for k in range(0,N):
    L_before += (np.log(w[0]*multivariate_normal.pdf(X[k], u1, cov1)) \
                +np.log(w[1]*multivariate_normal.pdf(X[k], u2, cov2) ) )

L_before = L_before / N
L_after = 0
iterations = 0
threshold = 1e-4

while np.abs(L_after - L_before) > threshold :

    iterations += 1
    L_before = L_after
    #for loop in range(0,5):
    # E Step
    for i in range(0,N):
        gamma1[i] = (w[0]*multivariate_normal.pdf(X[i], u1, cov1)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))


        gamma2[i] = (w[1]*multivariate_normal.pdf(X[i], u2, cov2)) / \
        (w[0]*multivariate_normal.pdf(X[i], u1, cov1) + w[1]*multivariate_normal.
 ↪pdf(X[i], u2, cov2))

    # Now, M Step
    u1_num = 0
    u2_num = 0
    for j in range(0,N):
        u1_num = u1_num + gamma1[j]*X[j]
        u2_num = u2_num + gamma2[j]*X[j]

    u1 = u1_num / np.sum(gamma1)
    u2 = u2_num / np.sum(gamma2)

    cov1_num = np.zeros([2,2])
    cov2_num = np.zeros([2,2])

    for k in range(0,N):
        cov1_num = cov1_num + gamma1[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u1, (-1,1)) )*(X[k] - u1)
        cov2_num = cov2_num + gamma2[k]*( np.reshape(X[k], (-1,1)) - np.
 ↪reshape(u2, (-1,1)) )*(X[k] - u2)

    cov1 = cov1_num / np.sum(gamma1)
    cov2 = cov2_num / np.sum(gamma2)

    w[0] = np.sum(gamma1)/N
    w[1] = np.sum(gamma2)/N

    L_after = 0
    for l in range(0,N):
        L_after = L_after + (np.log(w[0]*multivariate_normal.pdf(X[l], u1,␣
 ↪cov1)) \
            +np.log(w[1]*multivariate_normal.pdf(X[l], u2, cov2) ) )
    L_after = L_after / N

print("Number of Iterations: ", str(iterations))
print("Estimated mu1: ", str(u1))
print("\n")
print("Estimated mu2: ", str(u2))
print("\n")
print("Estimated w1: ", str(w[0]))
print("\n")
print("Estimated w2: ", str(w[1]))
print("\n")
print("Estimated sigma1:\n", str(cov1))
print("\n")
print("Estimated sigma2:\n", str(cov2))
```

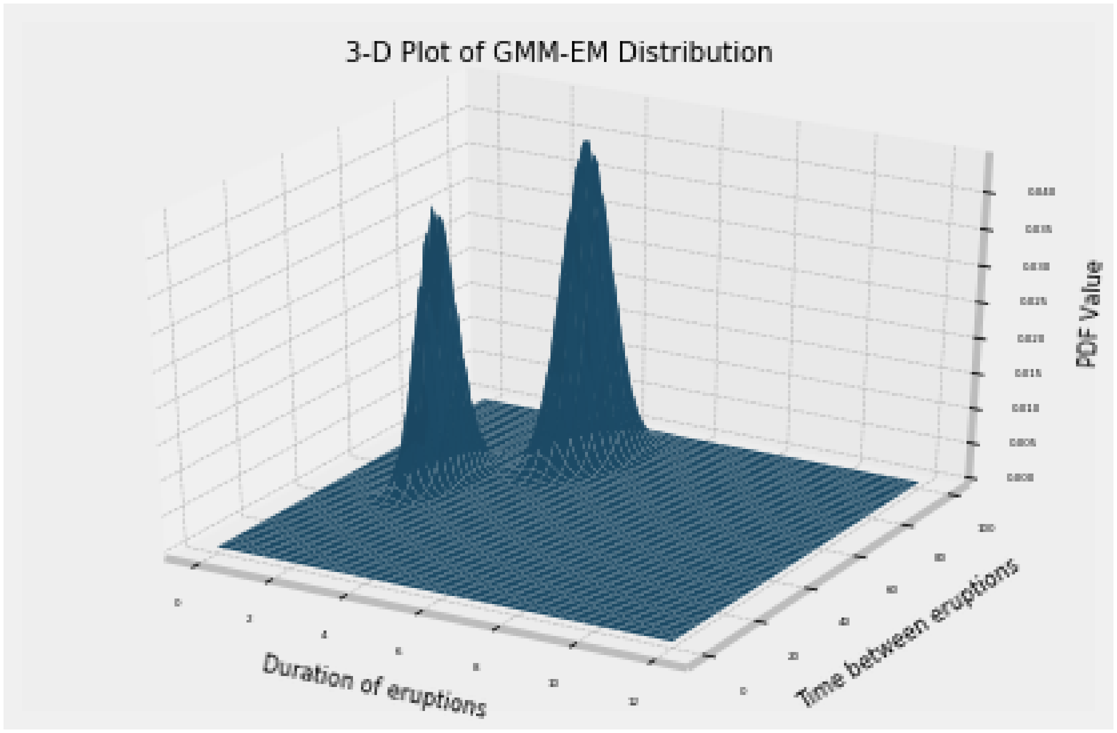
```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16: RuntimeWarning:
divide by zero encountered in log
  app.launch_new_instance()
```

```
Number of Iterations:  10
Estimated mu1:  [ 4.28966207 79.96811632]


Estimated mu2:  [ 2.03638856 54.47851745]


Estimated w1:  0.6441270990064238


Estimated w2:  0.35587290099357616


Estimated sigma1:
 [[ 0.16996832  0.94060779]
 [ 0.94060779 36.04619413]]


Estimated sigma2:
 [[ 0.06916776  0.43516851]
 [ 0.43516851 33.69728811]]
```

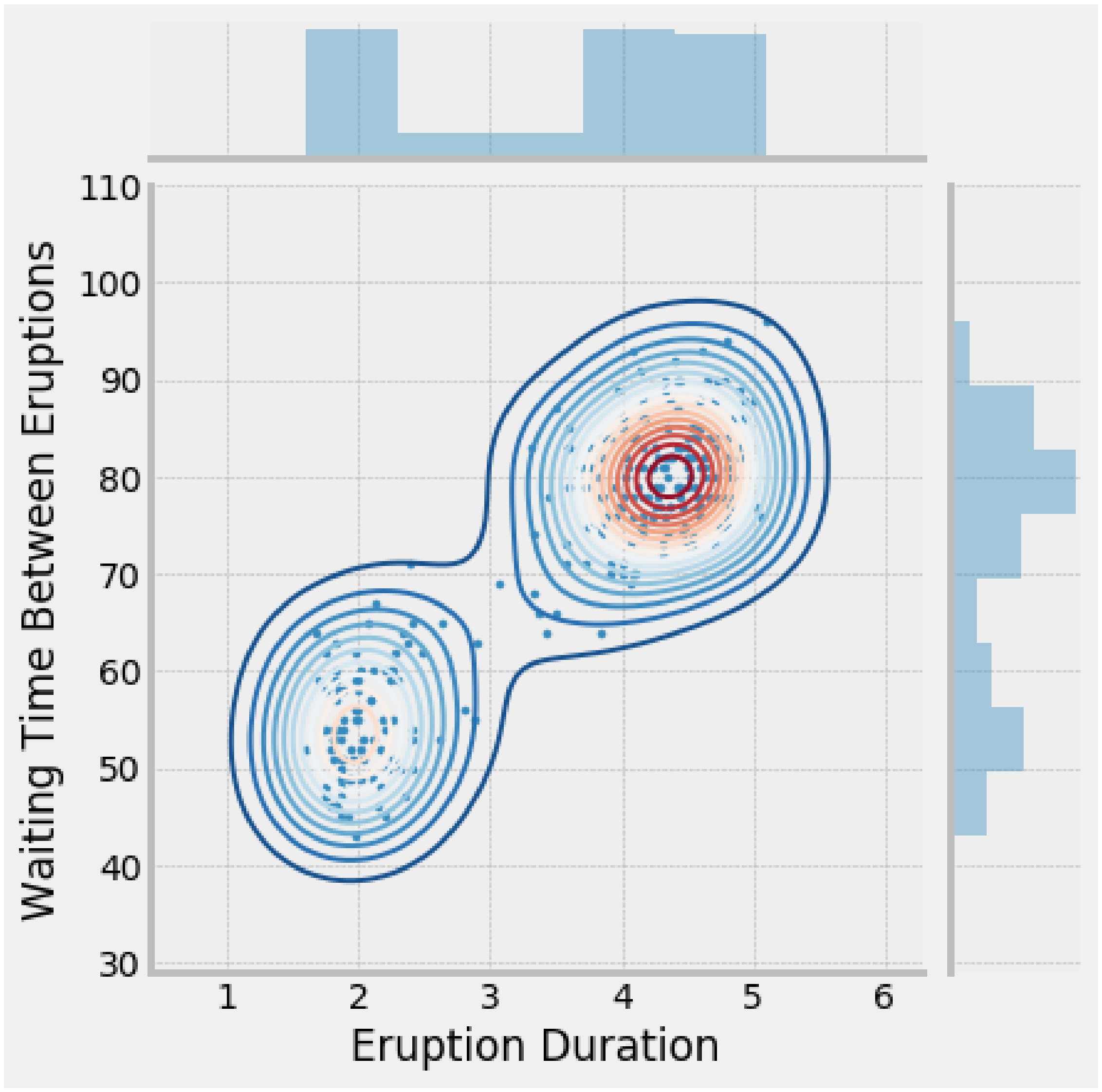
[25]:
```
plt.style.use('bmh')
x, y = np.mgrid[0:10:.01, 0:110:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x; pos[:, :, 1] = y

rv1 = multivariate_normal(u1, cov1)
rv2 = multivariate_normal(u2, cov2)
plt.contourf(x, y, rv2.pdf(pos)+rv1.pdf(pos))
#plt.scatter(X[:,0], X[:,1], c=kmeans.labels_.astype(float))
plt.xlabel("Duration of eruptions")
plt.ylabel("Time between eruptions")
plt.title('Countour of GMM-EM Distribution')
plt.tight_layout()
plt.figure(figsize=(8, 8))
plt.show()

x2, y2 = np.mgrid[0:12:120j, 0:100:10000j]

# Need an (N, 2) array of (x, y) pairs.
xy = np.column_stack([x2.flat, y2.flat])

z = w[0]*multivariate_normal.pdf(xy, mean=u1, cov=cov1) +␣
 ↪w[1]*multivariate_normal.pdf(xy, mean=u2, cov=cov2)
# Reshape back to the original grid.
z = z.reshape(x2.shape)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x2,y2,z)
# ax.plot_wireframe(x2,y2,z)
ax.set_xlabel("Duration of eruptions", fontsize = 8)
ax.set_ylabel("Time between eruptions", fontsize = 8)
ax.set_zlabel("PDF Value", fontsize = 8)
plt.title('3-D Plot of GMM-EM Distribution', fontsize = 10)
ax.tick_params(axis='both', which='major', labelsize=4)
ax.tick_params(axis='both', which='minor', labelsize=4)
plt.tight_layout()
plt.figure(figsize=(8, 8))
plt.style.use('bmh')
plt.show()

f = (sns.jointplot(X[:,0], X[:,1], marker='.') \
     .plot_joint(sns.kdeplot, n_levels=20, cmap="RdBu_r")) \
    .set_axis_labels('Eruption Duration', 'Waiting Time Between Eruptions')
```


Countour of GMM-EM Distribution
Time between eruptions
Duration of eruptions
0
20
40
60
80
100
0
2
4
6
8

```
<Figure size 576x576 with 0 Axes>
```


3-D Plot of GMM-EM Distribution
Duration of eruptions
Time between eruptions
PDF Value

```
<Figure size 576x576 with 0 Axes>
```



1.0.8 Question 4 Analysis

- The first plot is the contour plot of the GMM-EM Model we generated using the EM algorithm.
- The second plot, is a 3D plot which plots the joint distribution and allows for better visualization.
- The third plot is a contour plot but overlayed on the original data points.
- The K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The 'means' in the

K-means refers to averaging of the data, or finding the centroid.
- We can use the means we calculate from EM to then split the dataset into clusters.

```
[ ]:
```