Saad Mufti

CS 539, HW 1

Conceptual and Theoretical Questions

Q1 - Law of Total Variance / Expectation

Begin with the definition of the expectation function:

$$E[X] = \sum_x x \cdot p(X = x)$$

From here, the conditional expectation is logical

$$E_X[X|Y] = \sum_x x \cdot p(X = x|Y = y)$$

So,

$$E_Y[E_X[X|Y]] = \sum_y p(Y = y) \left[\sum_x x \cdot p(X = x|Y = y) \right]$$

Putting $p(Y = y)$ together with $p(X = x|Y = y)$ results in the joint distribution of X and Y:

$$\sum_y \sum_x x \cdot p(X = x, Y = y)$$

Summing the joint distribution along y gives the marginal distribution of X:

$$\sum_x x \cdot p(X = x)$$

Which is the definition of $E[X]$

Because variance is defined in terms of the expectation function

$$\text{var}[X] = E[X^2] - E[X]^2$$

Then, using the first equation,

$$= E_Y[E_X[X^2|Y]] - E_Y[E_X[X|Y]]^2$$

Rewrite $E_X[X^2|Y]$ as $\text{var}_X[X|Y] + E_X[X|Y]^2$:

$$= E_Y[\text{var}_X[X|Y] + E_Y[E_X[X|Y]^2] - E_Y[E_X[X|Y]]^2$$

Finally, combine the last two terms to form variance

$$= E_Y[\text{var}_X[X|Y]] + \text{var}_Y[E_X[X|Y]]$$

Q2

The expectation of a random vector is simply a vector with its elements as the corresponding expected values of each component.

It is known that the expectation of the sum of two random variables is equal to the sum of their individual expectations:

$$E[X+Y] = \sum_x \sum_y (x+y) \cdot p(x,y)$$

Distribute:

$$E[X+Y] = \sum_x \sum_y x \cdot p(x,y) + \sum_x \sum_y y \cdot p(x,y)$$
$$E[X+Y] = \sum_x x \cdot p(x,y) + \sum_y y \cdot p(x,y)$$
$$E[X+Y] = E[X] + E[Y]$$

The above logic applies to random vectors as well, just on an element by element basis.

Covariance is defined as $\text{cov}(X,Y) = E[(X - E[X])(Y - E[Y])]$

Generalize to vectors where a random vector is $\sum_i X$:

$$\text{cov}(X,Y) = E[(\sum_i x_i - E[\sum_i x_i])(\sum_j y_j - E[\sum_j y_j])]$$

Rewrite to bring the sum operators out:

$$\text{cov}(X,Y) = \sum_{i,j} E[x_i y_j] - E[x_i]E[y_j]$$
$$\text{cov}(X,Y) = \sum_{i,j} \text{cov}(x_i, y_j)$$

Q3

The posterior distribution can be estimated as a product of the likelihood and prior $p(\mu)$ distributions.

The likelihood function is defined as the conditional probability of observing the dataset X given the model parameters. So,

$$L(X|\mu) = \prod_{i=1}^{N} N(x_i|\mu, \Sigma)$$

Thus, the posterior distribution is

$$p(\mu|X) = N(\mu|\mu_0, \Sigma_0) \cdot \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\Sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\Sigma^2}\right)$$

$$p(\mu|X) = N(\mu|\mu_0, \Sigma_0) \cdot (2\pi\Sigma^2)^{-N/2} \cdot \prod_{i=1}^{N} \exp\left(-\frac{(x_i-\mu)^2}{2\Sigma^2}\right)$$

$$p(\mu|X) = (2\pi\Sigma_0^2)^{-1/2} \exp\left(-\frac{(\mu-\mu_0)^2}{2\Sigma_0^2}\right) \left[-\frac{N}{2}\ln\left(2\pi\Sigma_0^2\right) - \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2\Sigma_0^2}\right]$$

Q4

The likelihood function $L(x|\mu,\lambda)$ is

$$\left(\frac{\lambda}{2\pi}\right)^{N/2}\prod_{i=1}^{N}\exp\left(-\frac{\lambda(x_i-\mu)^2}{2}\right)$$

Simplify, as in (2.152):

$$\left[\lambda^{\frac{1}{2}}\exp\left(-\frac{\lambda\mu^2}{2}\right)\right]^N\exp\left(\lambda\mu\sum_{i=1}^{N}x_i-\frac{\lambda}{2}\sum_{i=1}^{N}x_i^2\right)$$

As per (2.153), the prior can be expressed as:

$$\left[\lambda^{\frac{1}{2}}\exp\left(-\frac{\lambda\mu^2}{2}\right)\right]^\beta\exp\left(c\lambda\mu-d\lambda\right)$$

c, d, and β are constants. Multiplying the prior and likelihood gives:

$$\left[\lambda^{\frac{1}{2}}\exp\left(-\frac{\lambda\mu^2}{2}\right)\right]^{\beta+N}\exp\left[\lambda\left((c+\sum x_i)\mu+(d+\frac{1}{2}\sum x_i^2)\right)\right]$$

Since the posterior is in the same form as the prior, simply with new values given for c, d, and β, multiplying the prior and likelihood functions return a Gaussian and Gamma distribution.

Using the prior as a template, the posterior can be expressed with c_p, d_p, and β, using their prior values:

$$\begin{aligned}\beta_p&=\beta+N\\c_p&=c+\sum x_i\\d_p&=d+\sum x_i^2\end{aligned}$$

Q5 - Covariance of Independent RVs

Covariance is formally defined as

$$cov_{XY} = E[(X - \mu_X)(Y - \mu_Y)]$$

Where $E[X] = \mu_X$ and $E[Y] = \mu_Y$

If the definition above is expanded to

$$= E[XY - \mu_Y X - \mu_X Y + \mu_Y \mu_X]$$

Then it is possible to use the property of the expectation function, that $E[A_1 + A_2] = E[A_1] + E[A_2]$, which results in

$$= E[XY] - E[\mu_Y X] - E[\mu_X Y] + E[\mu_Y \mu_X]$$

Factor out the constants, as $E[aX] = aE[X]$

$$\begin{aligned} &= E[XY] - \mu_Y E[X] - \mu_X E[Y] + \mu_Y \mu_X \\ &= E[XY] - \mu_X \mu_Y - \mu_X \mu_Y + \mu_Y \mu_X \\ &= E[XY] - \mu_X \mu_Y \end{aligned}$$

Finally, notice that because X and Y are independent, then $E[XY] = E[X]E[Y]$, resulting in

$$\mu_X \mu_Y - \mu_X \mu_Y = 0$$

Q6 - Kullback-Leibler Divergence

Kullback-Leiber Divergence is defined as:

$$KL(p||q) = -\int p(x) \ln q(x) dx + \int p(x) \ln p(x) dx$$

Or, combining the integrals:

$$KL(p||q) = -\int p(x) \ln \frac{q(x)}{p(x)} dx$$

The Gaussian distribution is mathematically expressed as

$$N(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

And its natural logarithm (for ease of calculation later):

$$\ln N(\mu, \sigma) = \frac{1}{2}\ln 2\pi - \frac{(x-\mu)^2}{2\sigma^2}$$

Evaluating the KL Divergence:

$$\begin{aligned}
&= -\int p(x) \left[\frac{1}{2}\ln 2\pi - \frac{(x-m)^2}{2s^2}\right] dx + \int p(x) \left[\frac{1}{2}\ln 2\pi - \frac{(x-\mu)^2}{2\sigma^2}\right] dx \\
&= -\int p(x) \left[-\frac{1}{2}\ln 2\pi - \frac{(x-m)^2}{2s^2}\right] dx - \frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2} \\
&= \frac{1}{2}\ln\left(2\pi s^2\right) + \int p(x) \frac{(x-m)^2}{2s^2} dx - \frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2} \\
&= \frac{1}{2}\ln\left(2\pi s^2\right) + \frac{1}{2s^2}\int p(x)(x^2 - 2mx + m^2) dx - \frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2} \\
&= \frac{1}{2}\ln\left(2\pi s^2\right) + \frac{1}{2s^2}\left[\int x^2 \cdot p(x) dx - 2m\int x \cdot p(x) dx + m^2 \int p(x) dx\right] - \frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2}
\end{aligned}$$

Use $E[g(x)] = \int g(x)p(x)dx$:

$$= \frac{1}{2}\ln\left(2\pi s^2\right) + \frac{1}{2s^2}\left[E[X^2] - 2m \cdot E[X] + m^2\right] - \frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2}$$

Use $E[X^2] = \text{var}[X] + E[X]^2 = \sigma^2 + \mu^2$

$$\begin{aligned}
&= \frac{1}{2}\ln\left(2\pi s^2\right) + \frac{1}{2s^2}\left[\sigma^2 + \mu^2 - 2m\mu + m^2\right] - \frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2} \\
&= \frac{1}{2}\ln\left(2\pi s^2\right) + \frac{1}{2s^2}\left[\sigma^2 + (\mu - m)^2\right] - \frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2}
\end{aligned}$$

Finally, simplify:

$$\ln\left(\frac{s}{\sigma}\right) + \frac{\sigma^2}{2s^2} + \frac{(\mu-m)^2}{2s^2} - \frac{1}{2}$$

Q7

Part a

The expectation of a continuous RV is $E[X] = \int x \cdot p(x) dx$

Apply to the given inverse gamma function:

$$E[X] = \frac{\beta^\alpha}{\Gamma(\alpha)} \int_0^\infty x^{-\alpha} \exp\left(-\frac{\beta}{x}\right) dx$$

$$E[X] = \frac{\beta^\alpha}{\Gamma(\alpha)} \cdot \frac{\Gamma(\alpha - 1)}{\beta^{\alpha-1}}$$

$$E[X] = \frac{\beta}{\Gamma(\alpha)} \cdot \Gamma(\alpha - 1)$$

$$E[X] = \frac{\beta}{\alpha - 1}$$

Part b

Here, the likelihood function is $L(\alpha, \beta) = \prod_{i=1}^N p(x_i | \alpha, \beta)$:

Change the product to a summation by taking the log of the likelihood function:

$$\ln\left(L(\alpha, \beta)\right) = \sum_{i=1}^N \ln \frac{\beta^\alpha}{\Gamma(\alpha)} + \ln x_i^{-\alpha-1} - \frac{\beta}{x_i}$$

$$\ln\left(L(\alpha, \beta)\right) = N \cdot \ln \frac{\beta^\alpha}{\Gamma(\alpha)} + \sum_{i=1}^N \ln x_i^{-\alpha-1} - \frac{\beta}{x_i}$$

Maximize β with $\frac{\partial L}{\partial \beta} = 0$

$$\frac{\partial L}{\partial \beta} = \frac{N\alpha}{\beta} - \sum_{i=1}^N \frac{1}{x_i} = 0$$

$$\beta_{ML} = \frac{N\alpha}{\sum_{i=1}^N x_i^{-1}}$$

Maximize α with $\frac{\partial L}{\partial \alpha} = 0$

$$\frac{\partial L}{\partial \alpha} = N \cdot \ln \frac{\beta^\alpha}{\Gamma(\alpha)} + \sum_{i=1}^N \ln x_i^{-\alpha-1} - \frac{\beta}{x_i}$$

$$0 = N \frac{\partial}{\partial \alpha} \left(\ln \frac{\beta^\alpha}{\Gamma(\alpha)} \right) + \sum_{i=1}^N \frac{\partial}{\partial \alpha} \ln x_i^{-\alpha-1}$$

Solve for α to find α_{ML}

Part c

A distribution belongs in the exponential distribution family if it follows the following form:

$$p(x|\theta) = h(x) g(\theta) \exp\left(\theta^{\mathrm{T}} u(x)\right)$$

$f(x)$ fits this condition, where $g(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}$ and $h(x) = x^{-\alpha-1}$

Part d, e

Y becomes a gamma distribution with the given parameters, resulting in a PDF of:

$$p(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp(-\beta x)$$

Using the given parameters:

$$p(x) = ce^{-cx}$$

As this is still of the exponential distribution family, a proper prior would be the gamma distribution

$$\Gamma(c|a,b)$$

Where $a = 1, b = 2$

The posterior becomes

$$p(c|x_i) = \Gamma(c|a=1,b=2)c^N \sum_i^N e^{-cx_i}$$

For N observations

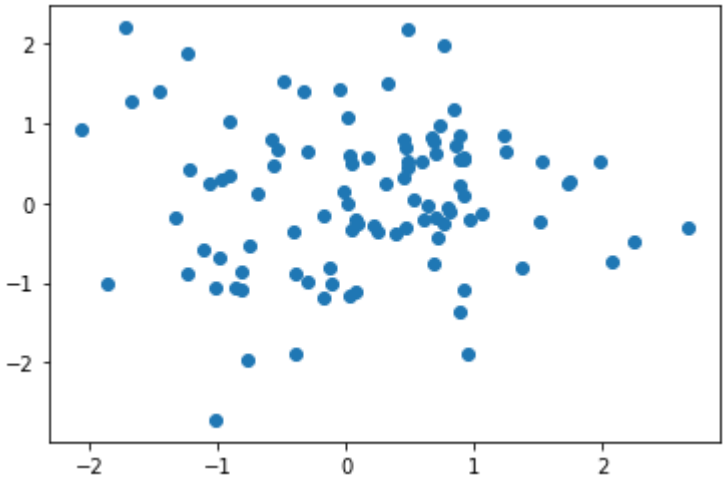
Application Questions

In [ ]:
```
# Using Python 3.8.1

# Imports
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy import stats
from sklearn.metrics import RocCurveDisplay
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
import random
import math
```

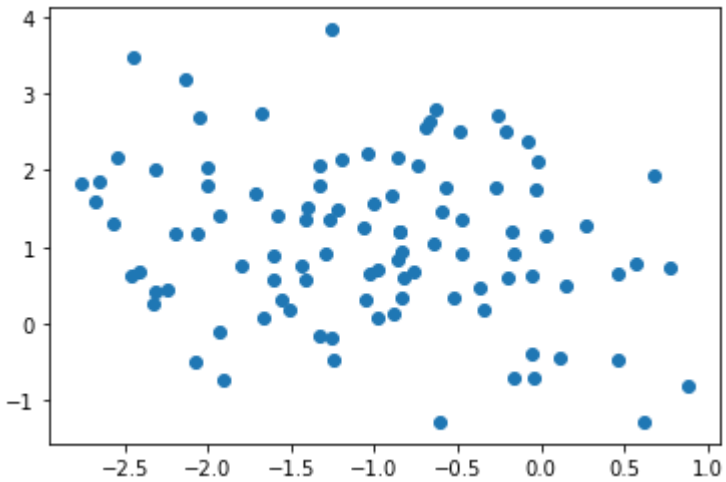
1. Sampling from a Distribution (15 pts)

In [ ]:
```
# Part a
mv_gauss_a = np.random.multivariate_normal(
                          mean=[0,0],
                          cov=[[1,0],[0,1]],
                          size=100)
plt.figure(0)
plt.scatter([x[0] for x in mv_gauss_a], [y[1] for y in mv_gauss_a])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x1c1139029a0>
```

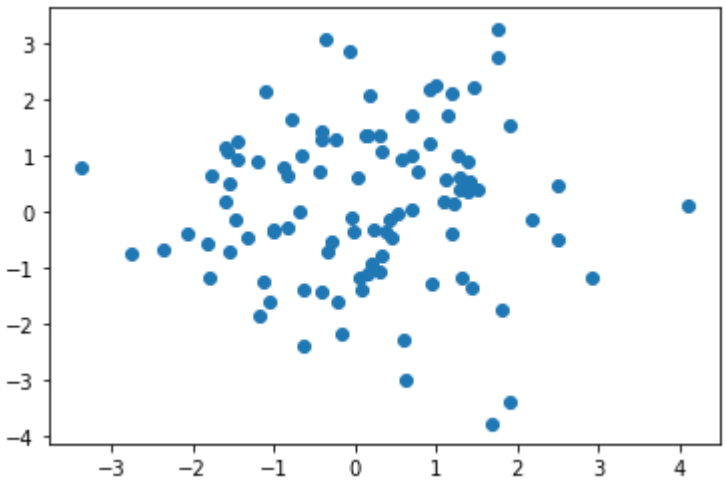


In [ ]:
```
# Part b
mv_gauss_b = np.random.multivariate_normal(
                  mean=[-1,1],
                  cov=[[1,0],[0,1]],
                  size=100)
plt.figure(1)
plt.scatter([x[0] for x in mv_gauss_b], [y[1] for y in mv_gauss_b])

_="""
Because only the mean changed, the multivariate gaussian's shape (e.g. "skinni
ness") does not change, but the location of its peak does. Now the points are
 no longer clustered around (0,0), but are now further up and to the left.
"""
```



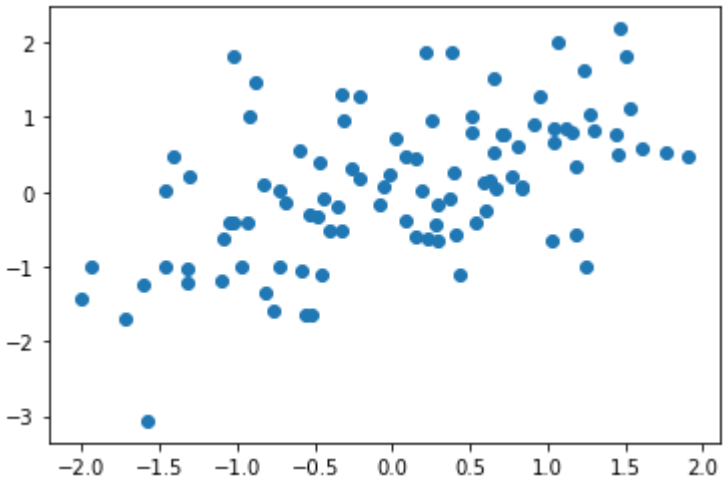
In [ ]:
```
# Part c
mv_gauss_c = np.random.multivariate_normal(
                mean=[0,0],
                cov=[[2,0],[0,2]],
                size=100)
plt.figure(2)
plt.scatter([x[0] for x in mv_gauss_c], [y[1] for y in mv_gauss_c])

"""
It is evident that increasing the variance of x1 and x2 increases the spread a
cross both axes.
"""
```

Out[ ]:
```
'\nIt is evident that increasing the variance of x1 and x2 increases the spre
ad across both axes. \n'
```



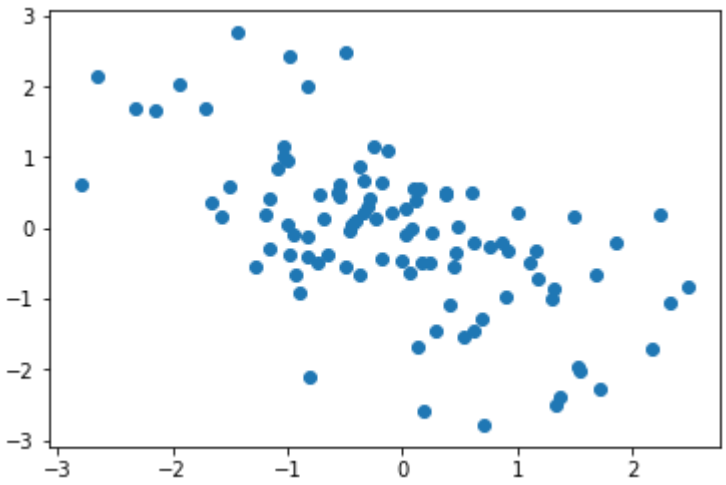
In [ ]:
```
# Part d
mv_gauss_d = np.random.multivariate_normal(
                mean=[0,0],
                cov=[[1,0.5],[0.5,1]],
                size=100)
plt.figure(3)
plt.scatter([x[0] for x in mv_gauss_d], [y[1] for y in mv_gauss_d])

"""
The distribution of the samples now resembles a line with a positive slope
"""
```

Out[ ]:
```
'\nThe distribution of the samples now resembles a line with a positive slope
\n'
```



In [ ]:
```
# Part e
mv_gauss_e = np.random.multivariate_normal(
                mean=[0,0],
                cov=[[1,-0.5],[-0.5,1]],
                size=100)
plt.figure(4)
plt.scatter([x[0] for x in mv_gauss_e], [y[1] for y in mv_gauss_e])

"""
The distribution of the samples now resembles a line with a negative slope
"""
```

Out[ ]:
```
'\nThe distribution of the samples now resembles a line with a negative slope
\n'
```

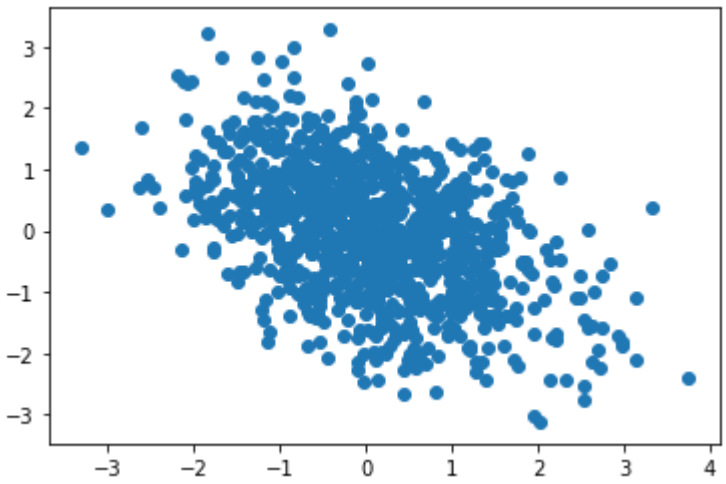


In [ ]:
```
# Part f

cov_matrix = np.cov(m=mv_gauss_e, rowvar=False)
av = (np.average([x[0] for x in mv_gauss_e]),
        np.average([y[1] for y in mv_gauss_e]))

print("Covariance matrix:\n", cov_matrix)
print("Average:\n", av)
```

```
Covariance matrix:
 [[ 1.2127543  -0.72850594]
 [-0.72850594  1.24061739]]
Average:
 (-0.07355973165570365, -0.08993759472706986)
```

In [ ]:
```
# Part g
mv_gauss_g = np.random.multivariate_normal(
                mean=[0,0],
                cov=[[1,-0.5],[-0.5,1]],
                size=1000)

plt.figure(5)
plt.scatter([x[0] for x in mv_gauss_g], [y[1] for y in mv_gauss_g])
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x1c1139a54f0>
```



In [ ]:
```
# Part h
cov_matrix_h = np.cov(m=mv_gauss_g, rowvar=False)
mean_h = (np.average([x[0] for x in mv_gauss_g]),
        np.average([y[1] for y in mv_gauss_g]))

print("Covariance matrix:", cov_matrix_h)
print("Mean", mean_h)
```

```
Covariance matrix: [[ 1.08662591 -0.50165445]
 [-0.50165445  1.06746058]]
Mean (0.060964514776955955, -0.026922190823540415)
```

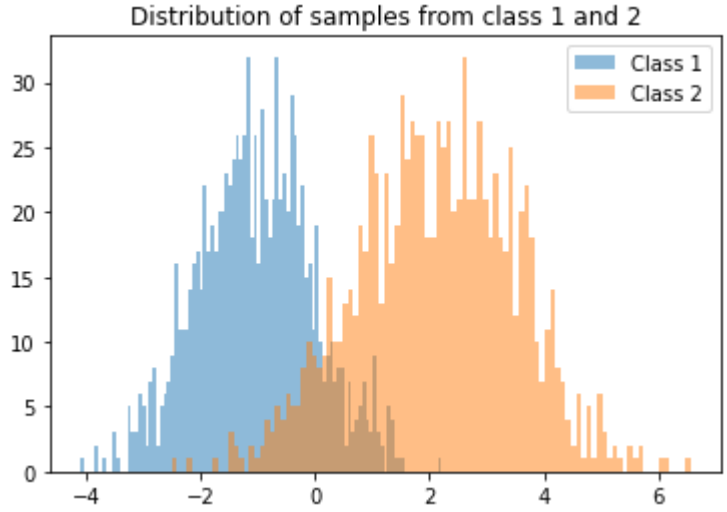
In [ ]:
```
# Part i, j
gaussian = lambda samples: np.random.multivariate_normal(
                                mean=[0,0],
                                cov=[[1,-0.5],[-0.5,1]],
                                size=samples)
def gaussian_average(sampling_size):
    data = gaussian(sampling_size)
    mean = (np.average([x[0] for x in data]),
            np.average([y[1] for y in data]))
    print("Average for sampling size = {:<2}:".format(sampling_size),
            mean)

def gaussian_cov(sampling_size):
    data = gaussian(sampling_size)
    cov = np.cov(data, rowvar=False)
    print("Cov(x1,x2), sample size = " + str(sampling_size) + ":{:<35}".format
(cov[0][1]))

gaussian_average(10)
gaussian_average(100)
gaussian_average(1000)

gaussian_cov(10)
gaussian_cov(100)
gaussian_cov(1000)


_="""
As the sampling size increases, the mean and the covariance both approach the
 parameters given to the model in g), and are more consistent as the distribut
ion is sampled multiple times. With a sample size of 10, the covariance and me
an are very volatile, jumping around as they are sampled multiple times. At a
 sampling size of 1000, these values vary much less. It is possible for an est
imate with a smaller sampling size to be as accurate compared to that of one w
ith a higher sampling size, but it will not be as likely or conssitent
"""
```

```
Average for sampling size = 10: (0.202329068815451, -0.5301533705299037)
Average for sampling size = 100: (-0.02809361651484865, 0.12531132990124305)
Average for sampling size = 1000: (0.05145237148377632, -0.0674404753671594)
Cov(x1,x2), sample size = 10:-0.12617159471219894
Cov(x1,x2), sample size = 100:-0.5427987874001927
Cov(x1,x2), sample size = 1000:-0.5102361321055116
```

2. Bayes Classifier (15 pts)

In [ ]:
```
# Part a, b
def class_init(samples_a, samples_b):
    class_1 = np.random.normal(loc=-1, scale=1, size=samples_a)
    class_2 = np.random.normal(loc=2, scale=np.sqrt(2), size=samples_b)
    return class_1, class_2
```

In [ ]:
```
# Part c
class_1_b, class_2_b = class_init(1000, 1000)
def part_c_hist(class_1, class_2):
    plt.figure(10)
    plt.hist(x=class_1, bins=100, alpha=0.5, label="Class 1")
    plt.hist(x=class_2, bins=100, alpha=0.5, label="Class 2")
    plt.title("Distribution of samples from class 1 and 2")
    plt.legend()
part_c_hist(class_1_b, class_2_b)
```

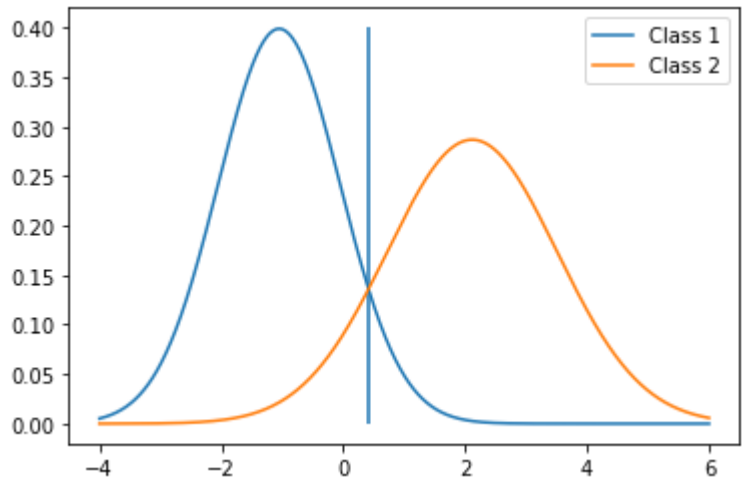


In [ ]:
```python
# Part d Using MLE to estimate distribution parameters

def obj(params, data):
    mean, st_dev = params
    # Logarithms make the likelihood function easier
    # to work with, as discussed in class. Although
    # MLE maximizes the likelihood function, minimizing
    # the negative likelihood function is also equivalent.
    neg_log_likelihood = -np.sum(stats.norm.logpdf(
                                    data,
                                    loc=mean,
                                    scale=st_dev))
    return neg_log_likelihood

def calc_mle(class_1, class_2):
    init_guess = np.array([0, 1])

    # MLE - Class 1
    mean_1, st_dev_1 = minimize(
                        lambda params: obj(params, class_1),
                        x0=init_guess, method="Powell").x
    print(f"Class 1 - Mean={mean_1}, var={st_dev_1**2}")

    # MLE - Class 2
    mean_2, st_dev_2 = minimize(
                        lambda params: obj(params, class_2),
                        x0=init_guess, method="Powell").x
    print(f"Class 2 - Mean={mean_2}, var={st_dev_2**2}")

    # The scipy docs advise to use Powell or Nelder-Mead
    # method for noisy data, which is indeed the case.
    return [(mean_1, st_dev_1), (mean_2, st_dev_2)]
est_stats_b = calc_mle(class_1_b, class_2_b)
```

```
Class 1 - Mean=-1.0528562718551424, var=1.0007035501395347
Class 2 - Mean=2.1141778963763302, var=1.9351940597506507
```

In [ ]:
```
# Part e
def plot_mle_pdf(est_stats):
    mean_1, st_dev_1 = est_stats[0]
    mean_2, st_dev_2 = est_stats[1]
    plt.figure(11)
    pdf_x = np.linspace(-4,6,num=1000)
    mle_pdf_1 = stats.norm.pdf(x=pdf_x, loc=mean_1, scale=st_dev_1)
    mle_pdf_2 = stats.norm.pdf(x=pdf_x, loc=mean_2, scale=st_dev_2)
    plt.plot(pdf_x, mle_pdf_1, label="Class 1")
    plt.plot(pdf_x, mle_pdf_2, label="Class 2")

    ints = np.argwhere(np.diff(np.sign(mle_pdf_1 - mle_pdf_2)))
    plt.vlines(pdf_x[ints], ymin=0, ymax=0.4)
    plt.legend()
    print(f"Decision boundary = {pdf_x[ints][0][0]}, classify as class 1 if be
low this value and class 2 if above")
    return pdf_x[ints[0][0]]
bound_b = plot_mle_pdf(est_stats_b)
_="""
The decision boundary is simple: it indicates a change in which probability of
the two distributions is higher.
"""
```

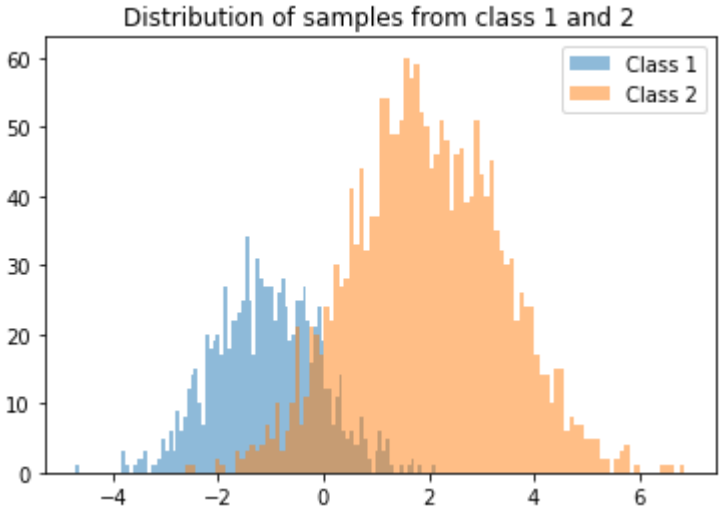
```
Decision boundary = 0.4144144144144146, classify as class 1 if below this val
ue and class 2 if above
```

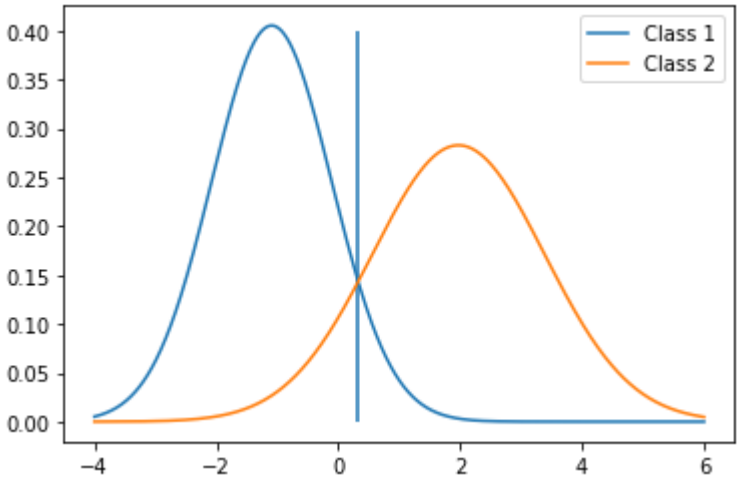


In [ ]:
```
# Part g
class_1_ub, class_2_ub = class_init(1000, 2000)
```

In [ ]:
```
part_c_hist(class_1_ub, class_2_ub)
est_stats_ub = calc_mle(class_1_ub, class_2_ub)
bound_ub = plot_mle_pdf(est_stats_ub)
```

```
Class 1 - Mean=-1.0927148762043832, var=0.9694407673810834
Class 2 - Mean=1.9716067823721273, var=1.9904796077492188
Decision boundary = 0.32432432432432456, classify as class 1 if below this va
lue and class 2 if above
```

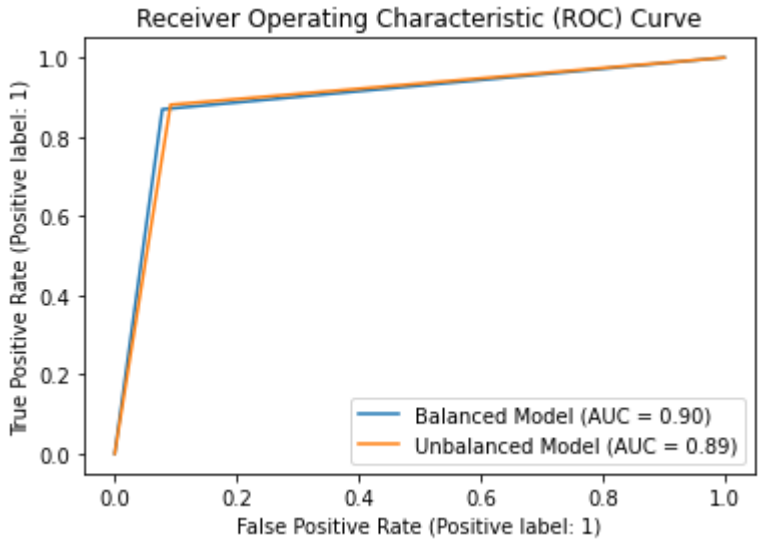




In [ ]:
```
# Part h, i
class_1_tests = np.random.normal(loc=-1, scale=1, size=2000)
class_2_tests = np.random.normal(loc=2, scale=np.sqrt(2), size=2000)
test_set = np.concatenate((class_1_tests, class_2_tests))
# Ground truth labels
gt_labels = np.array([0] * 2000 + [1] * 2000)

predict_b = np.array([1 if i > bound_b else 0 for i in test_set])
predict_ub = [1 if i > bound_ub else 0 for i in test_set]

error_b = [1 if predict_b[i] == gt_labels[i] else 0 for i in range(len(test_se
t))]
error_ub = [1 if predict_ub[i] == gt_labels[i] else 0 for i in range(len(test_
set))]
print("Balanced model error rate: ", 1 - np.average(error_b))
print("Unbalanced model error rate: ", 1 - np.average(error_ub))

fig_roc, ax = plt.subplots()
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")

RocCurveDisplay.from_predictions(y_true=gt_labels, y_pred=predict_b, ax=ax, na
me="Balanced Model")
RocCurveDisplay.from_predictions(y_true=gt_labels, y_pred=predict_ub, ax=ax, n
ame="Unbalanced Model")
```

```
Balanced model error rate:  0.10424999999999995
Unbalanced model error rate:  0.10524999999999995
```

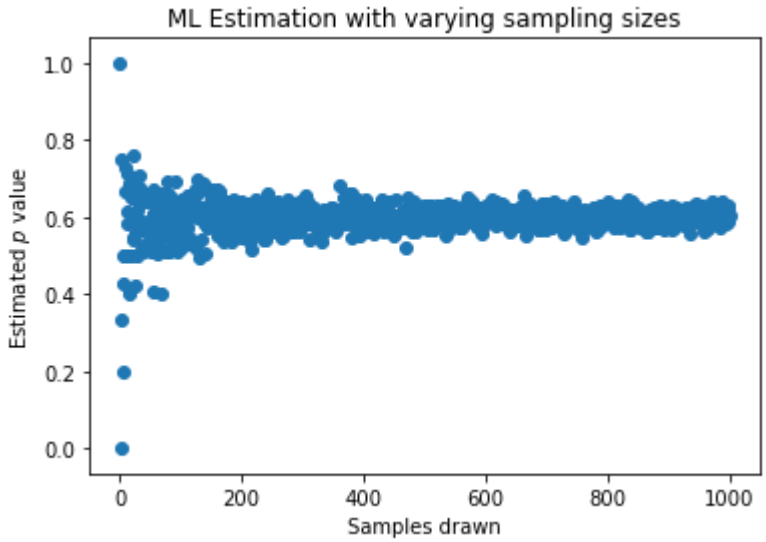
Out[ ]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21cbc4665b0>
```



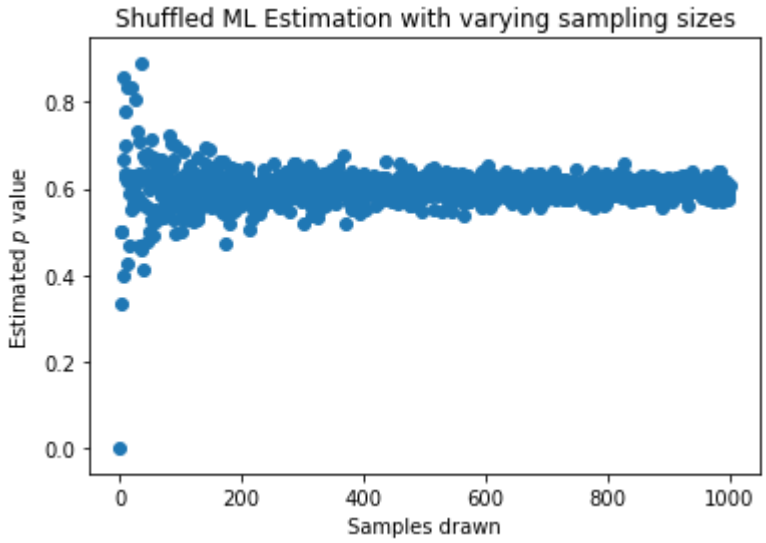
3. MLE, MAP, Bernoulli Distribution, and Beta Prior (15 pts)

In [ ]:
```
# Part a, b
bernoulli = []
def bernoulli_mle(samples):
    bernoulli = np.random.binomial(n=1, p=0.6, size=samples)
    b_guess = 0.5
    def obj_bernoulli(p, data):
        return -np.sum(stats.bernoulli.logpmf(data, p=p))

    p_est = minimize(lambda v:
                     obj_bernoulli(p=v, data=bernoulli),
                     x0=b_guess, method="Nelder-Mead").x
    return p_est

bern_samples_a = np.random.binomial(n=1, p=0.6, size=1000)
sample_nums = np.arange(start=1, stop=1001, step=1)
bernoulli_samples = np.array([bernoulli_mle(i) for i in sample_nums])
plt.figure(300)
plt.scatter(sample_nums, bernoulli_samples)
plt.xlabel("Samples drawn")
plt.ylabel("Estimated $p$ value")
plt.title("ML Estimation with varying sampling sizes")
```

Out[ ]:
```
Text(0.5, 1.0, 'ML Estimation with varying sampling sizes')
```

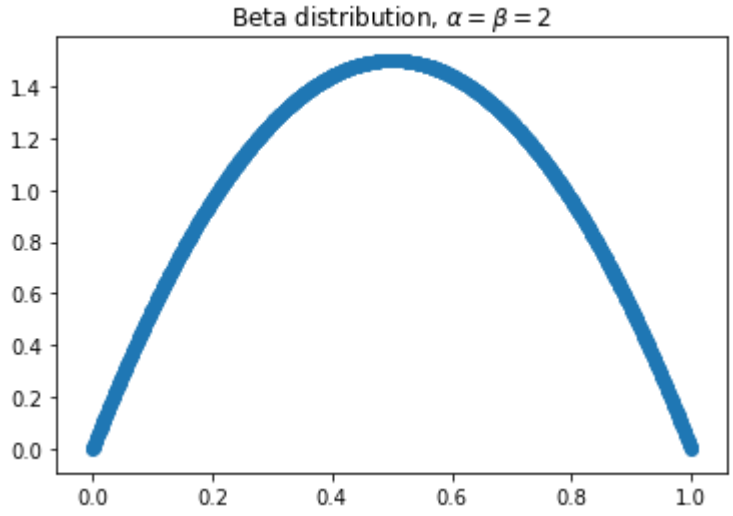


In [ ]:
```
# Part c
random.shuffle(bernoulli)
bernoulli_samples = np.array([bernoulli_mle(i) for i in sample_nums])
plt.figure(301)
plt.scatter(sample_nums, bernoulli_samples)
plt.xlabel("Samples drawn")
plt.ylabel("Estimated $p$ value")
plt.title("Shuffled ML Estimation with varying sampling sizes")

_="""
When the sampling size is very small, the estimated values for p vary greatly
 and nearly span the entire range of p. However, as the sampling size increase
s, the estimated value for p quickly reduces in variability and stays close to
its actual value, 0.6. There does not appear to be any noticeable difference b
etween the results of parts b and c, with the only change in procedure being t
he shuffling of the dataset, which is logical.
"""
```



In [ ]:
```
# Part d
beta_x = np.linspace(0, 1.0, num=1000)
beta = stats.beta.pdf(x=beta_x, a=2, b=2)
plt.figure(302)
plt.scatter(beta_x, beta)
plt.title(r"Beta distribution, $\alpha=\beta=2$")
```

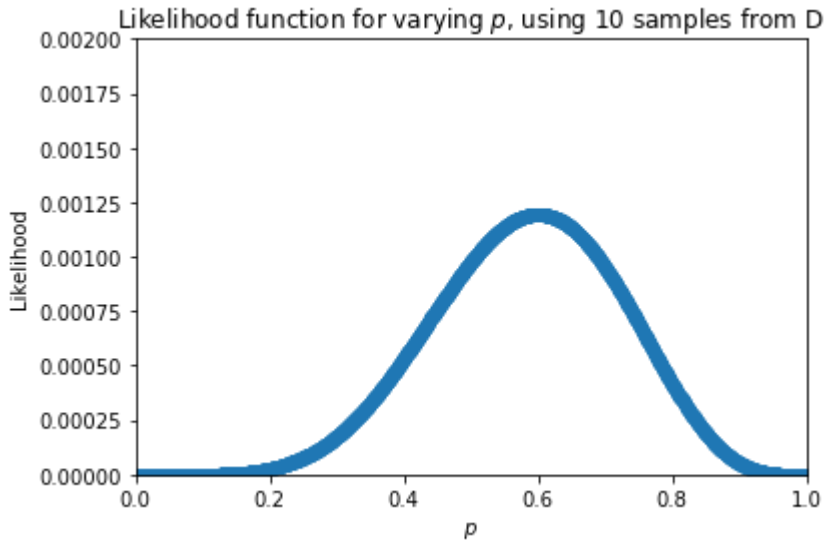
Out[ ]:
```
Text(0.5, 1.0, 'Beta distribution, $\\alpha=\\beta=2$')
```



In [ ]:
```
# Part e - plotting likelihood vs p for 10 samples

p_binned = np.linspace(0, 1, num=1000)
def likelihood_gen(samples):
    part_e_samples = bern_samples_a[:samples]
    def bern_likelihood(p):
        return math.prod(
            [p**x_i * (1 - p)**(1-x_i)
                for x_i in part_e_samples]
        )

    likelihood = [bern_likelihood(p) for p in p_binned]
    print("Max probability", max(likelihood))
    return likelihood

plt.figure(303)
plt.scatter(p_binned, likelihood_gen(10))
plt.axis([0,1,0,0.002])
plt.title("Likelihood function for varying $p$, using 10 samples from D")
plt.ylabel("Likelihood")
plt.xlabel("$p$")
```

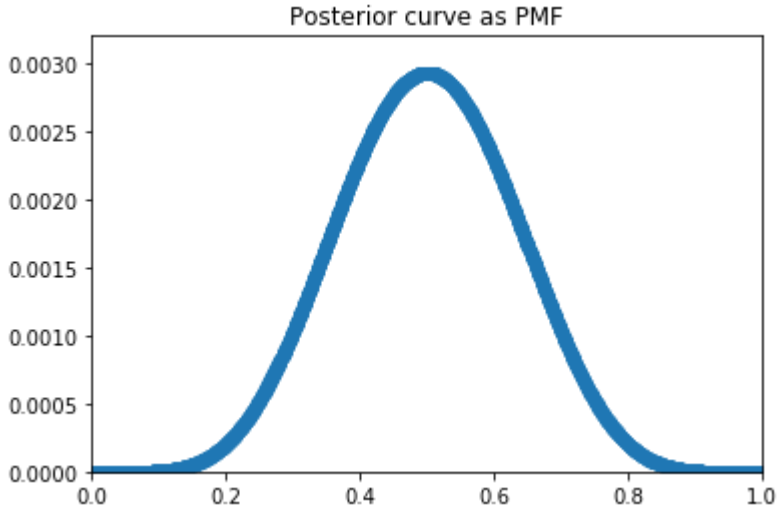
```
Max probability 0.001194389611605917
```

Out[ ]:
```
Text(0.5, 0, '$p$')
```



In [ ]:
```
# Part f - generating posterior curve
plt.figure(304)
plt.axis([0, 1, 0, .0032])
plt.title("Posterior curve as PMF")
posterior = np.multiply(beta, likelihood_gen(10))

print("sum before", np.sum(posterior))
# Normalize the curve for part g, so that sum(posterior) = 1
posterior = np.multiply(posterior, 2)
# posterior = np.divide(posterior,
#                           np.sum(posterior))
                         # np.trapz(posterior, beta_x))
print("sum", np.sum(posterior))
print("max pt", max(posterior))
plt.scatter(beta_x, posterior)

_="""
The addition of the prior distribution made the resulting curve thinner and ta
ller compared to the likelihood function itself, which indicates a higher cert
ainty of the model's estimation.
"""

# Part g - MAP estimation (mode of posterior)
p_mode = beta_x[list(posterior).index(max(posterior))]
print("MAP Estimation of p: ", p_mode)

# Part h
def expec(moment=1, d=posterior[posterior != 0]):
    return np.sum(
                [(p_binned[i]**moment) * d[i] for i in range(len(d))])
variance = lambda d: expec(moment=2, d=d) - expec(d=d)**2
print("Variance of p: ", variance(posterior))
```

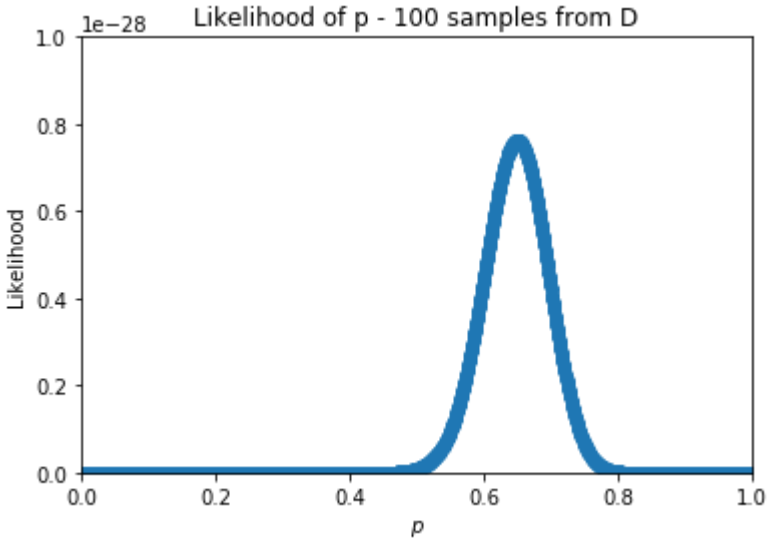
```
Max probability 0.0009765576074170125
sum before 0.499000999000999
sum 0.998001998001998
max pt 0.0029296698867100684
MAP Estimation of p:  0.4994994994994995
Variance of p:  0.017131869129871163
```

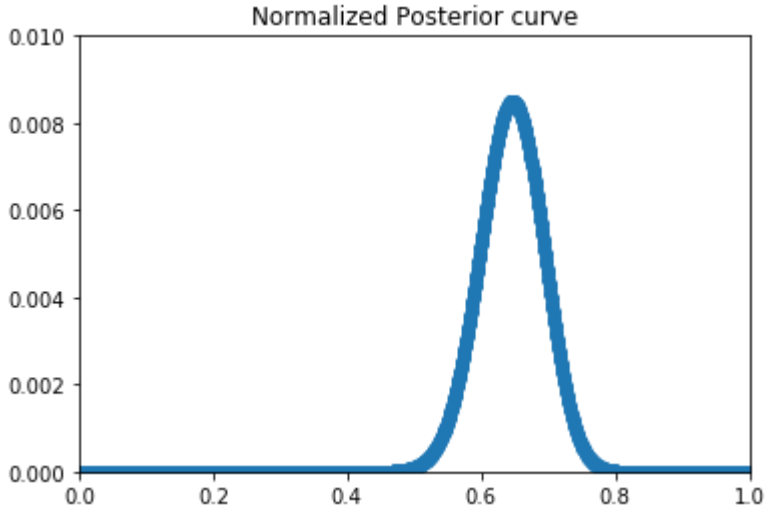


In [ ]:

```
# Part i
plt.figure(305)
plt.axis([0,1,0,1E-28])
plt.scatter(p_binned, likelihood_gen(100))
# print(likelihood_gen(100))
plt.title("Likelihood of p - 100 samples from D")
plt.xlabel("$p$")
plt.ylabel("Likelihood")

plt.figure(306)
plt.axis([0, 1, 0, 0.01])
plt.title("Normalized Posterior curve")
posterior = np.multiply(beta, likelihood_gen(100))
print(np.sum(posterior))
posterior = np.divide(posterior,
                      np.sum(posterior))
                      # np.trapz(posterior, beta_x))

plt.scatter(beta_x, posterior)
p_mode = beta_x[list(posterior).index(max(posterior))]
print("MAP Estimation of p: ", p_mode)
print("Variance of p: ", variance(posterior))
```

```
Max probability 7.61619441740692e-29
Max probability 7.61619441740692e-29
1.2256420492255538e-26
MAP Estimation of p:  0.6466466466466466
Variance of p:  0.0021828331924485522
```

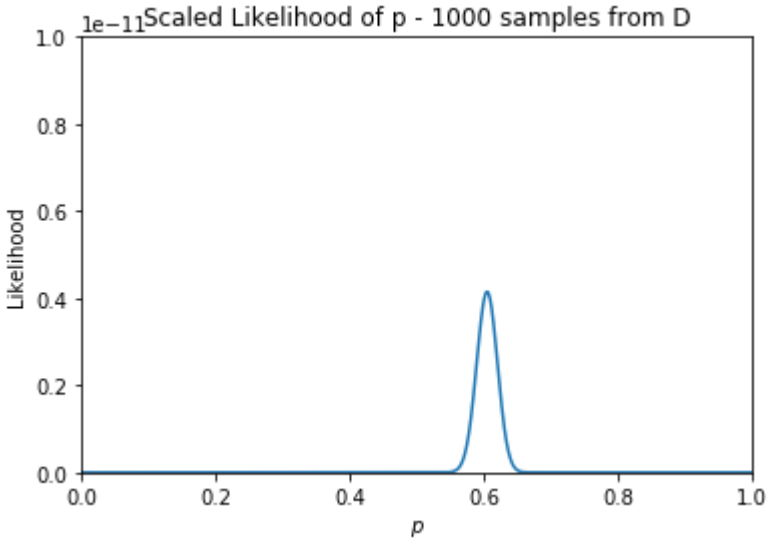


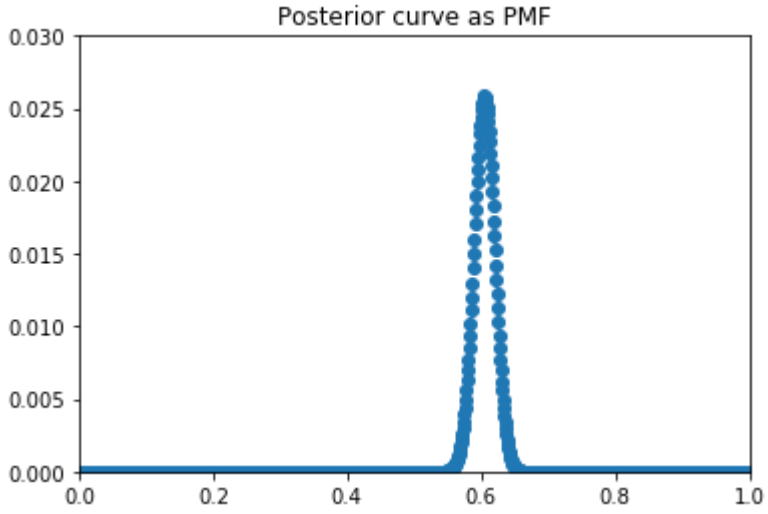


In [ ]:

```
# Part j
plt.figure(307)
plt.axis([0, 1, 0, 1E-11])
plt.plot(p_binned, np.multiply(likelihood_gen(1000), 1E280))
plt.title("Scaled Likelihood of p - 1000 samples from D")
plt.xlabel("$p$")
plt.ylabel("Likelihood")

plt.figure(308)
plt.axis([0, 1, 0, 0.03])
plt.title("Posterior curve as PMF")
posterior = np.multiply(beta, likelihood_gen(1000))
posterior = np.divide(posterior,
                      np.sum(posterior))
                      # np.trapz(posterior, beta_x))

plt.scatter(beta_x, posterior)
p_mode = beta_x[list(posterior).index(max(posterior))]
print("MAP Estimation of p: ", p_mode)
print("Variance of p: ", variance(posterior))

_="""
As the number of samples from the distribution increased, the likelihood and p
osterior curves became much thinner, indicating the model is more certain abou
t the likelihood of what p is. Numerically this is shown by the variance decre
asing by about a factor of 10 between each run. The MAP estimate of p also imp
roves, with the estimated value using 1000 samples having a lower error compar
ed to that of 100 samples compared to that of 10 samples.
"""
```

```
Max probability 4.146793529948664e-292
Max probability 4.146793529948664e-292
MAP Estimation of p:  0.6046046046046046
Variance of p:  0.0002378733070729222
```

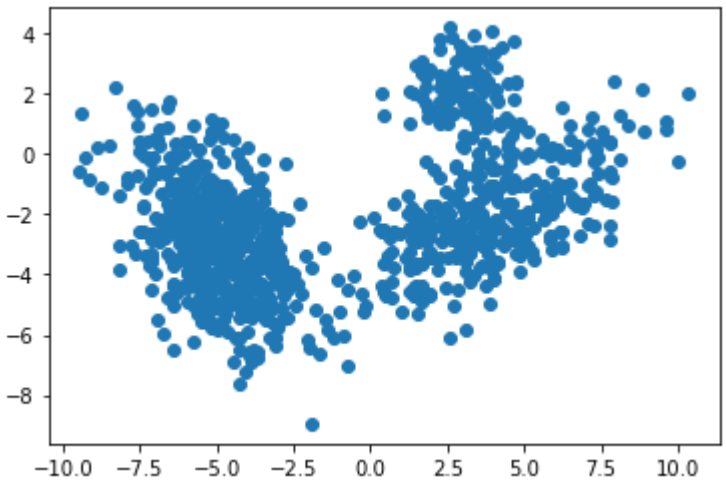




4. Gaussian Mixture Model (15 pts)

In [ ]:
```
# Part a
gm = GaussianMixture(n_components=3,covariance_type='full')

weights = np.array([0.1, 0.6, 0.3])
mu = np.array([[3,2], [-5,-3], [4,-2]])
covariances = np.array([
    [[1,0],[0,1]],
    [[2,-1],[-1,3]],
    [[6,3],[3,3]]
])

gm.weights_ = weights
gm.means_ = mu
gm.covariances_ = covariances
data, labels_actual = gm.sample(1000)
gm_x, gm_y = data.transpose()

plt.figure(400)
plt.scatter(gm_x, gm_y)
```

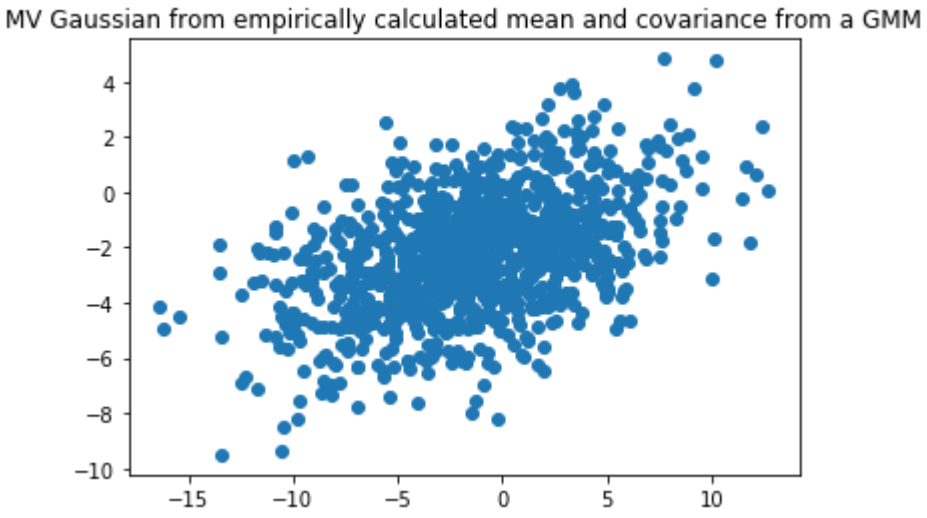
Out[ ]:
```
<matplotlib.collections.PathCollection at 0x1c125a8a1c0>
```



In [ ]:
```
# Part b - calculated mean, covariance
emp_cov = np.cov(m=data.transpose())
emp_mean = (np.average(gm_x), np.average(gm_y))
print("Empirical Covariance", emp_cov)
print("Empriical mean", emp_mean)
```

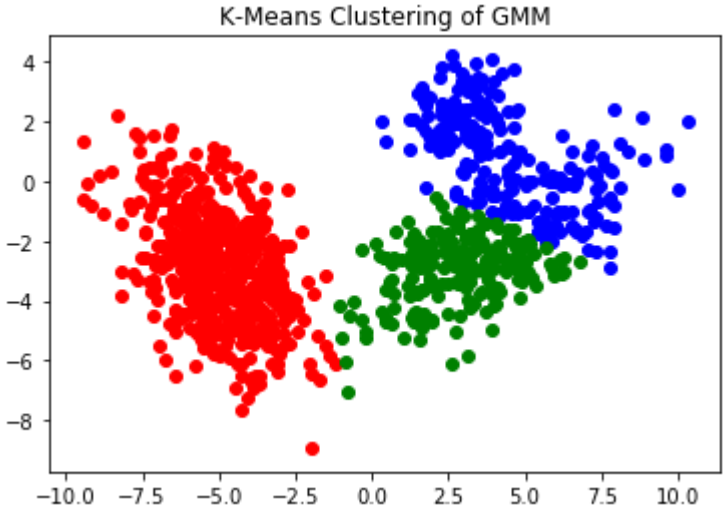
```
Empirical Covariance [[21.72112132  4.86851449]
 [ 4.86851449  4.90657213]]
Empriical mean (-1.3887208286203578, -2.2131131171768033)
```

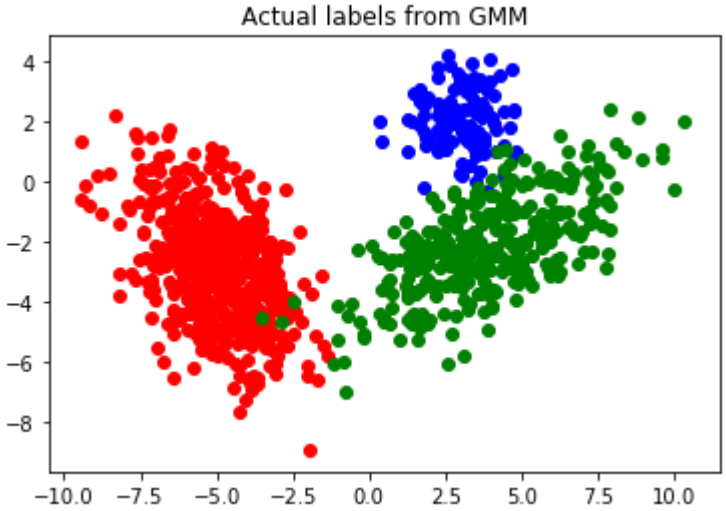
In [ ]:
```
# Part c, d
emp_gauss = np.random.multivariate_normal(
                          mean=emp_mean,
                          cov=emp_cov,
                          size=1000)
plt.figure(401)
plt.scatter(emp_gauss[:,0], emp_gauss[:,1])
plt.title("MV Gaussian from empirically calculated mean and covariance from a
 GMM")

_="""
While the range of the mv gaussian does appear to match that of the GMM, the m
ean of the GMM was sparsely populated with clusters nearby but not directly on
the mean. The MV gaussian on the other hand is centered around its mean (by de
finition). The GMM has three relatively distinct groupings of samples, while t
he MV gaussian is one large group, as expected since it is only one gaussian d
istribution.
"""
```



In [ ]:

```
# Part e - Using K-Means Clustering
alg = KMeans(n_clusters=3)
pred_labels = alg.fit_predict(data)

plt.figure(403)
plt.title("K-Means Clustering of GMM")

def color(label_val):
    if label_val == 0:
        return "blue"
    if label_val == 1:
        return "red"
    if label_val == 2:
        return "green"
for i in range(0,3):
    gm_x_n = [gm_x[x_i] for x_i in range(len(gm_x)) if pred_labels[x_i] == i]
    gm_y_n = [gm_y[y_i] for y_i in range(len(gm_y)) if pred_labels[y_i] == i]
    plt.scatter(gm_x_n, gm_y_n, color=color(i))

plt.figure(404)
plt.title("Actual labels from GMM")
for i in range(0,3):
    gm_x_n = [gm_x[x_i] for x_i in range(len(gm_x)) if labels_actual[x_i] == i
]
    gm_y_n = [gm_y[y_i] for y_i in range(len(gm_y)) if labels_actual[y_i] == i
]
    plt.scatter(gm_x_n, gm_y_n, color=color(i))


_="""
The K-Means clustering appears to work effectively with
clusters amply spaced from each other, such as the leftmost
group of points, but with the right two groups intersecting,
it mistakes the right half of component 3 for component 1.
"""
```


K-Means Clustering of GMM
4
2
0
−2
−4
−6
−8
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Actual labels from GMM
4
2
0
−2
−4
−6
−8
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

In [ ]: