```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io as sio

from collections import Counter
import random

from tabulate import tabulate
```

```
In [2]: data = sio.loadmat('mnist.mat')
```

Converting data from uint8 to float

```
In [3]: print("Type Before type-casting: "+str(type(data['trainX'][0][19])))

XTrain = data['trainX'].astype(float)
yTrain = data['trainY'][0].astype(float)

XTest = data['testX'].astype(float)
yTest = data['testY'][0].astype(float)

print("Type After  type-casting: "+str(type(XTrain[0][19])))
```

```
Type Before type-casting: <class 'numpy.uint8'>
Type After  type-casting: <class 'numpy.float64'>
```

```
In [4]: print("Shape of XTrain: "+str(np.shape(XTrain)))
print("Shape of yTrain: "+str(np.shape(yTrain)))

print("Shape of XTest: "+str(np.shape(XTest)))
print("Shape of yTest: "+str(np.shape(yTest)))
```

```
Shape of XTrain: (60000, 784)
Shape of yTrain: (60000,)
Shape of XTest: (10000, 784)
Shape of yTest: (10000,)
```

Helper Functions

Uniformly Random Dataset

```
In [5]: def getDataMatrix(X,y,m):
    index = np.random.choice(X.shape[0], m, replace=False)
    x_random = X[index]
    y_random = y[index]
    return x_random, y_random
```

Euclidean Distance

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

```
In [6]: ## Helper function to finds the euclidean distance between 2 images: x, y
def getEuclideanDistance(x, y):
    return np.sqrt(sum((x - y) ** 2))
```

Majority Voting System

Chose winner by random guess if it's a tie

```
In [7]: # Finds the majority class/label out of the given labels
def majorityVotingSystem(all_possible_labels):
    # Counter gives the frequency count
    counter = Counter(all_possible_labels)
    # Finding the majority class.
    majority_count = max(counter.values())
    possibleAnswers = []
    for key, value in counter.items():
        if value == majority_count:
            # Add all possible candidates in the list
            possibleAnswers.append(key)
    # As we have 0 index, we are subtracting 1
    possibleAnswersLength = len(possibleAnswers)-1
    randomIndex = random.randint(0,possibleAnswersLength)
    return possibleAnswers[randomIndex]
```

PreComputed Distance Matrix

This matrix Sacrifices memory but saves computation (as told in the HW dexription)

```
In [8]: def getPreComputedDistanceMatrix(XTest, XTrain, yTrain, yTest):
    # PreComputed EuclideanDistance Matrix
    allDistances = []
    for indexValue,XTestImage in enumerate(XTest):
        # Distance of 1 test image from every point in the training set
        # Format: (distance, training Image Label)
        distanceFromAllTrainImages = [(getEuclideanDistance(XTestImage, XTrainImage), XTrainLabel, inde
x, yTest[indexValue])
                            for index, (XTrainImage, XTrainLabel) in enumerate(zip(XTrain, yTrain))]

        # Dimension for each test Image would be: ( m * 2)
        # m -> no of training images
        # 2 -> EuclideanDistance and the training image label
        allDistances.append(distanceFromAllTrainImages)

    # overaall shape: n * m * 2
    # n -> no of test images
    # m -> no of training labels
    # 2 -> EuclideanDistance and the training image label

    # Assertions to confirm all matrixes are of expected size
    allDistancesShape = np.shape(allDistances)
    assert (allDistancesShape[0] == np.shape(XTest)[0]),  f"Expected:{np.shape(XTest)[0]} Found: {allDi
stancesShape[0]}"
    assert (allDistancesShape[1] == np.shape(XTrain)[0]), f"Expected:{np.shape(XTrain)[0]} Found: {allD
istancesShape[1]}"
    assert (allDistancesShape[2] == 4)                  , f"Expected:{4} Found: {allDistancesShape[2]}"

    return allDistances
```

```
In [9]: def predict(k, train_images, train_labels, test_images, distances):
    # sort the distances list by distances
    sortedDistances = sorted(distances, key=lambda x:x[0])
    # extract only k closest labels
    k_labels = [label for (_, label,_,_) in sortedDistances[:k]]
    # return the majority voted label
    return majorityVotingSystem(k_labels)
```

Q4 a) Accuracy for different values of m and k

```
In [10]: mValues = [10, 100, 1000, 10000]
# mValues = [10]

kValues = [1,2,3,4,5,6,7,8,9,10]
# kValues = [1]


for m in mValues:
    results = []
    train_images, train_labels = getDataMatrix(XTrain,yTrain,m)

    test_images = XTest
    test_labels = yTest

    allDistances = getPreComputedDistanceMatrix(test_images, train_images, train_labels, test_labels)

    for k in kValues:
        total_correct = 0
        for index,test_image in enumerate(test_images):
            pred = predict(k, train_images, train_labels, test_image,allDistances[index])
            if pred == test_labels[index]:
                total_correct += 1

        acc = (total_correct / (index+1)) * 100
        results.append([m,k,acc])

    columns = ['m','k', 'Accuracy']
    df = pd.DataFrame(results, columns=columns)
    print(tabulate(df, headers='keys', tablefmt='psql'))
```

	m	k	Accuracy
0	10	1	36.25
1	10	2	27.05
2	10	3	23.53
3	10	4	22.25
4	10	5	21.34
5	10	6	21.04
6	10	7	19.98
7	10	8	18.62
8	10	9	16.62
9	10	10	9.84

	m	k	Accuracy
0	100	1	71.03
1	100	2	65.29
2	100	3	65.97
3	100	4	64.53
4	100	5	62.68
5	100	6	60.72
6	100	7	59.5
7	100	8	58.41
8	100	9	56.98
9	100	10	55.84

	m	k	Accuracy
0	1000	1	88.18
1	1000	2	85.72
2	1000	3	87.72
3	1000	4	87.76
4	1000	5	87.63
5	1000	6	87.47
6	1000	7	87.47
7	1000	8	87.1
8	1000	9	87.02
9	1000	10	86.74

	m	k	Accuracy
0	10000	1	95.3
1	10000	2	94.31
2	10000	3	95.16
3	10000	4	94.99
4	10000	5	94.84
5	10000	6	94.81
6	10000	7	94.7
7	10000	8	94.67
8	10000	9	94.49
9	10000	10	94.43

Q4 a) Final Comments

As the value of m increases, the accuracy increased and error rate decreased.

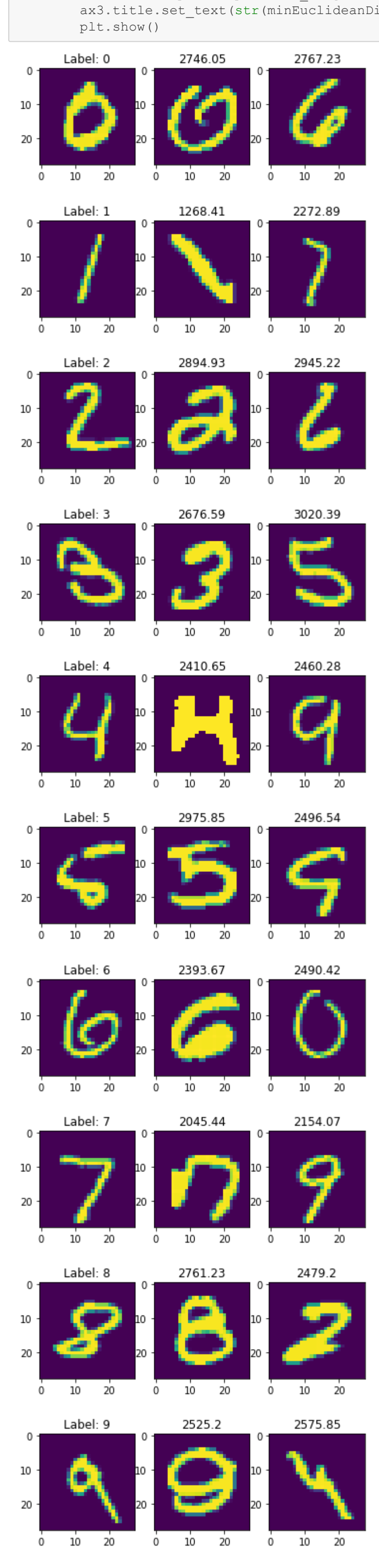
The increase in accuracy against the number of samples after a certain number is not very high, i.e. from m= 1000 to m=10000, we had around 8% increase in accurac.

The computation power (Time complexity) also increased with m.

After a certain m, the accuracy may not increase much (it may reach a threshold).

Q4 b) Analyzing results.

```
In [10]: mValues = [10000]


u, indices = np.unique(yTest, return_index=True)
test_images = XTest[indices]
test_labels = yTest[indices]

for m in mValues:

    train_images, train_labels = getDataMatrix(XTrain,yTrain,m)
    allDistances = getPreComputedDistanceMatrix(test_images, train_images, train_labels, test_labels)


    for index,value in enumerate(indices):
        temp = allDistances[index]

        maxArray = list(filter(lambda x: x[1] ==  x[3], temp))
        if len(maxArray) >0:
            a = sorted(maxArray, key=lambda x:x[0])
            maxIndex  = a[-1][2]
        else:
            continue

        minArray = list(filter(lambda x: x[1]!=  x[3], temp))
        if len(minArray) >0:
            b = sorted(minArray, key=lambda x:x[0])
            minXIndex = b[0][2]
        else:
            continue

        fig = plt.figure()
        minEuclideanDistance = round(minArray[0][0],2)
        maxEuclideanDistance = round(maxArray[-1][0],2)
        ax1 = fig.add_subplot(1,3,1)
        ax1.imshow(np.reshape(XTest[value],(28,28)))
        ax1.title.set_text('Label: '+str(int(test_labels[index])))
        ax2 = fig.add_subplot(1,3,2)
        ax2.imshow(np.reshape(train_images[maxIndex],(28,28)))
        ax2.title.set_text(str(maxEuclideanDistance))
        ax3 = fig.add_subplot(1,3,3)
        ax3.imshow(np.reshape(train_images[minXIndex],(28,28)))
        ax3.title.set_text(str(minEuclideanDistance))
        plt.show()
```



Q4b) Final Comments

Procedure followed

1) I have plotted the above graph for m = 100, i.e I have taken 100 random samples from the training set.

2) There are a total of 10,000 samples in the test data.

3) For each image in test set, we computed the distance from all the values in training set, i.e. we formed a 10,000 * 100 matrix.

4) We then computed the following (for 10 samples):

```
maximum distance from the digit in the training sample furthest from that digit but with the sam
e label.

minimum distance from the digit in the training sample closest to that digit but with the differ
ent label.
```

Conclusion

1) As you can see, for same labels which are most apart, the distance is mostly due to the fact they are tilted , i.e. in different orientation (or) different thickness.

2) For labels where there is a mismatch yet they are close is due to the fact, their orientation and thickness is almost same, i.e. most of the pixels look same and the difference between them is small, so we have a small euclidean distance but they have different label.

General Mismatches

0 -> 6

6 -> 0

This is classic error

4 -> 9

7 -> 9

9 -> 4

Q4c)

KNN isn't a great way to classify handwritten digits.

Reason:

1) Time Complexity:

As for each data in the test set, we have to compare it with every value in the training data.

This operation is very expensive and increases with the size of the training data.

2) Space Complexity:

If we store all the distances before, we can save some time during actual calculations.

We either have an enormous time complexity (or) space complexity.

KNN has a pretty good accuracy (95% when we have a training data size of 10,000) but it's relatively slow

Comparison with SVM or logistic regression

SVM is very similar to logistic regression but SVM has a stronger decision boundary, i.e. we have more confidence in our guesses.

Both SVM and logistic regression become good with more training data, as we try to minimize the loss, i.e we penalize a model if it's making a wrong guess and with more examples, it gets better.

Time complexity and space complexity is much lesser in both logistic regression and SVM as compared to KNN.

Final Conclusion:

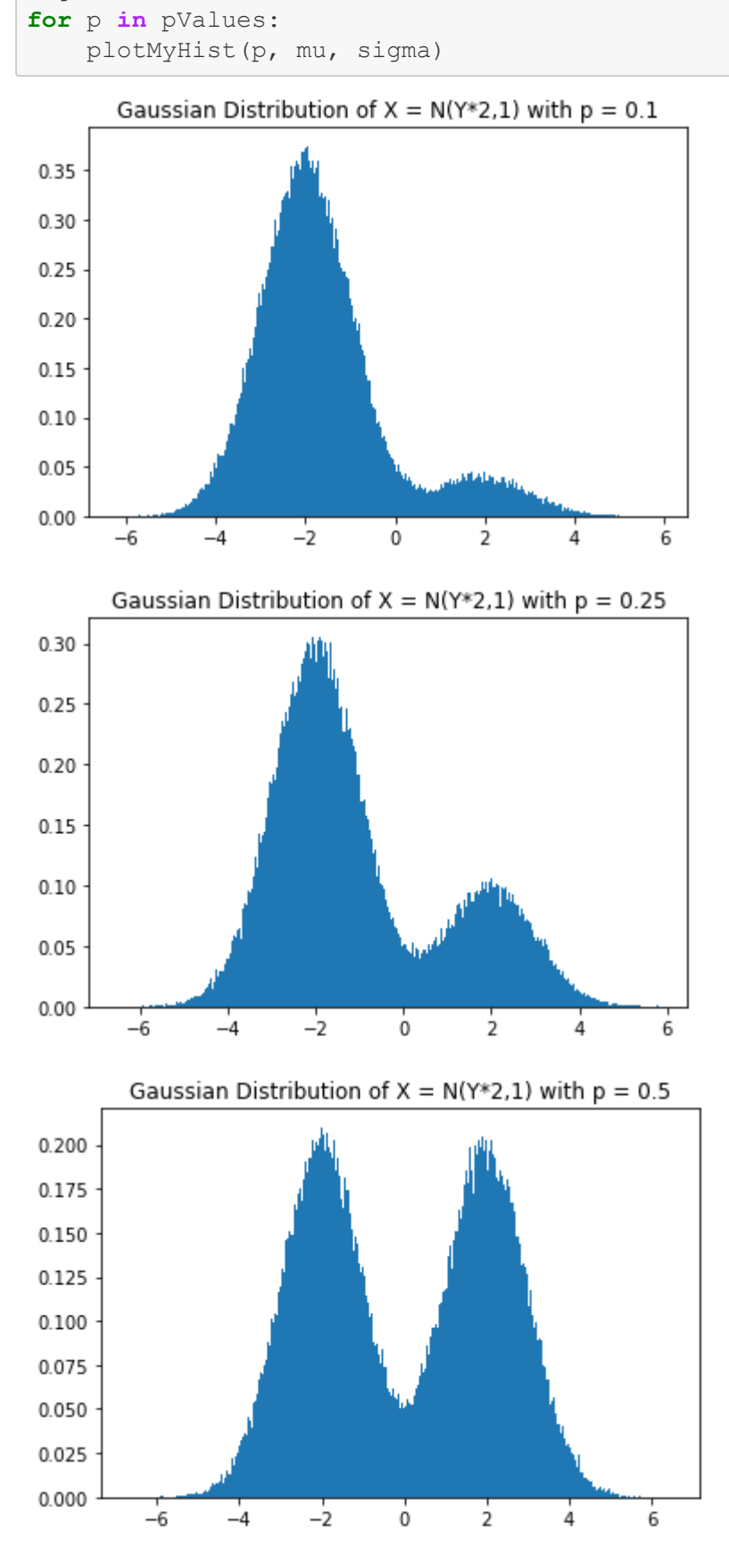
We can achieve a similar accuracy with logistic regression and SVM a s we did in KNN but with a much lesser time ans space complexity.

Challenge Question

```
In [11]: def plotMyHist(p, mu, sigma, m=1000):

    elements = [1,-1]
    probabilities = [p, 1-p]
    m =100000
    Y = np.random.choice(elements, m, p=probabilities)
    X = np.random.normal(Y * mu, sigma, m)
    count, bins, ignored = plt.hist(X, 300, density=True)
    plt.title("Gaussian Distribution of X = N(" + "Y*"+str(mu)+ ","+ str(sigma) +") with p = " + str(p
))
    plt.show()
    return
```

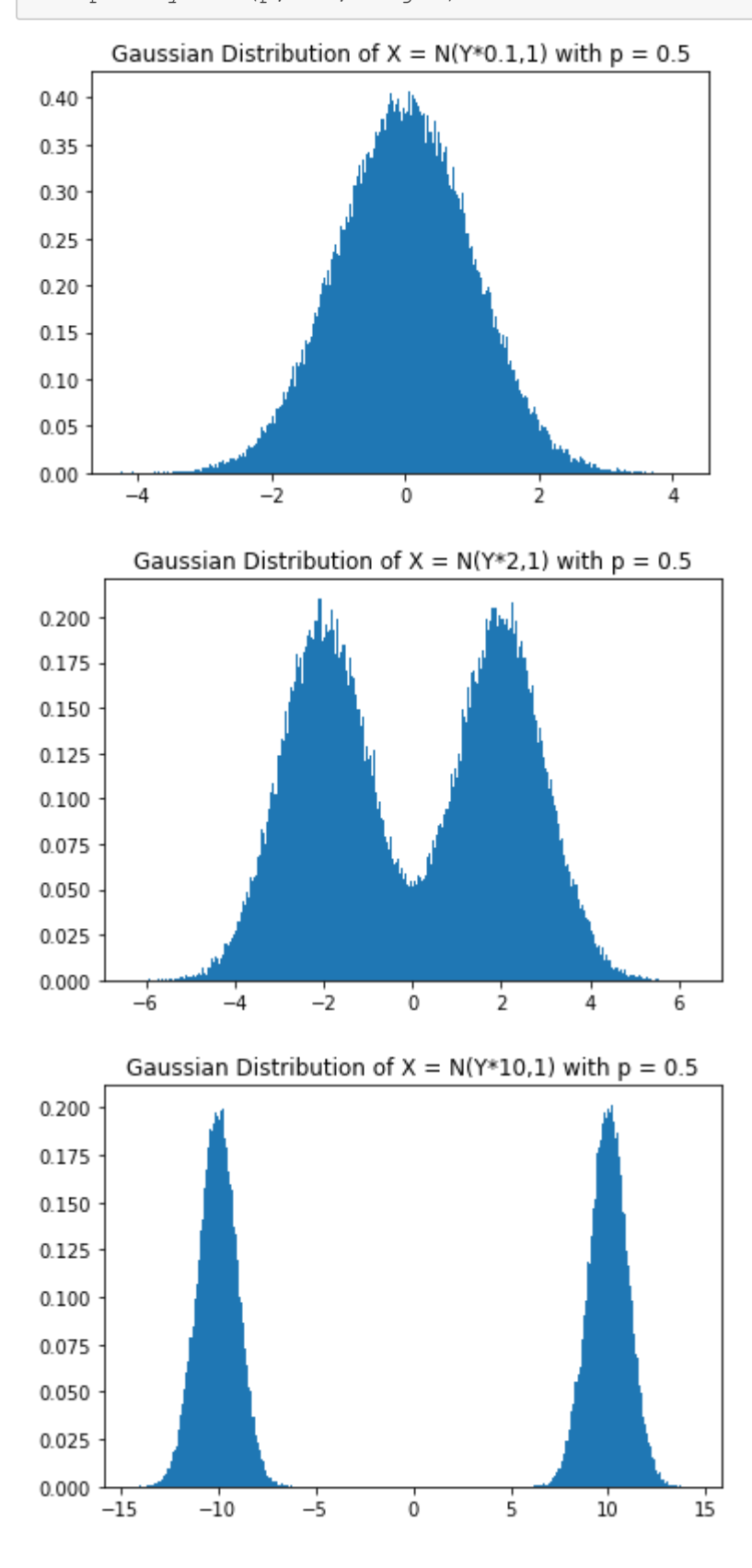
1a)

```
In [12]: pValues = [0.1, 0.25, 0.5]
mu = 2
sigma = 1
for p in pValues:
    plotMyHist(p, mu, sigma)
```



1b)

```
In [13]: muValues = [0.1,2,10]
p = 0.5
sigma = 1
for mu in muValues:
    plotMyHist(p, mu, sigma)
```



1c)

```
In [14]: sigmaValues = [0.1,1,3]
p = 0.5
mu = 2
for sigma in sigmaValues:
    plotMyHist(p, mu, sigma)
```