Homework 2

DUE NOV 11th at 11:59 PM

You are expected to turn in a **pdf version** of this notebook with all your **codes, results, and figures** (Use the print option). Make sure the figures and results are visible as you want them to appear in the pdf before turning it in. Please do not modify the instructions as doing so will limit our ability to follow and grade your answers.

Problem 1 (HW1 Problem 4)

In this problem, you will work on the clustering problem using Bottom-up Agglomerative clustering and K-mean clustering.

a) A 4-D dataset is given in **'iris.csv'** with the last column being the ground truth label. Load the file. Store the data in a variable ***X*** and store the label in a variable ***y***. Because clustering is an unsupervised task, there is no need for the labels during training.

In [191]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('iris.csv',header = None)
X = data[range(4)].values
y = data[4].values
```

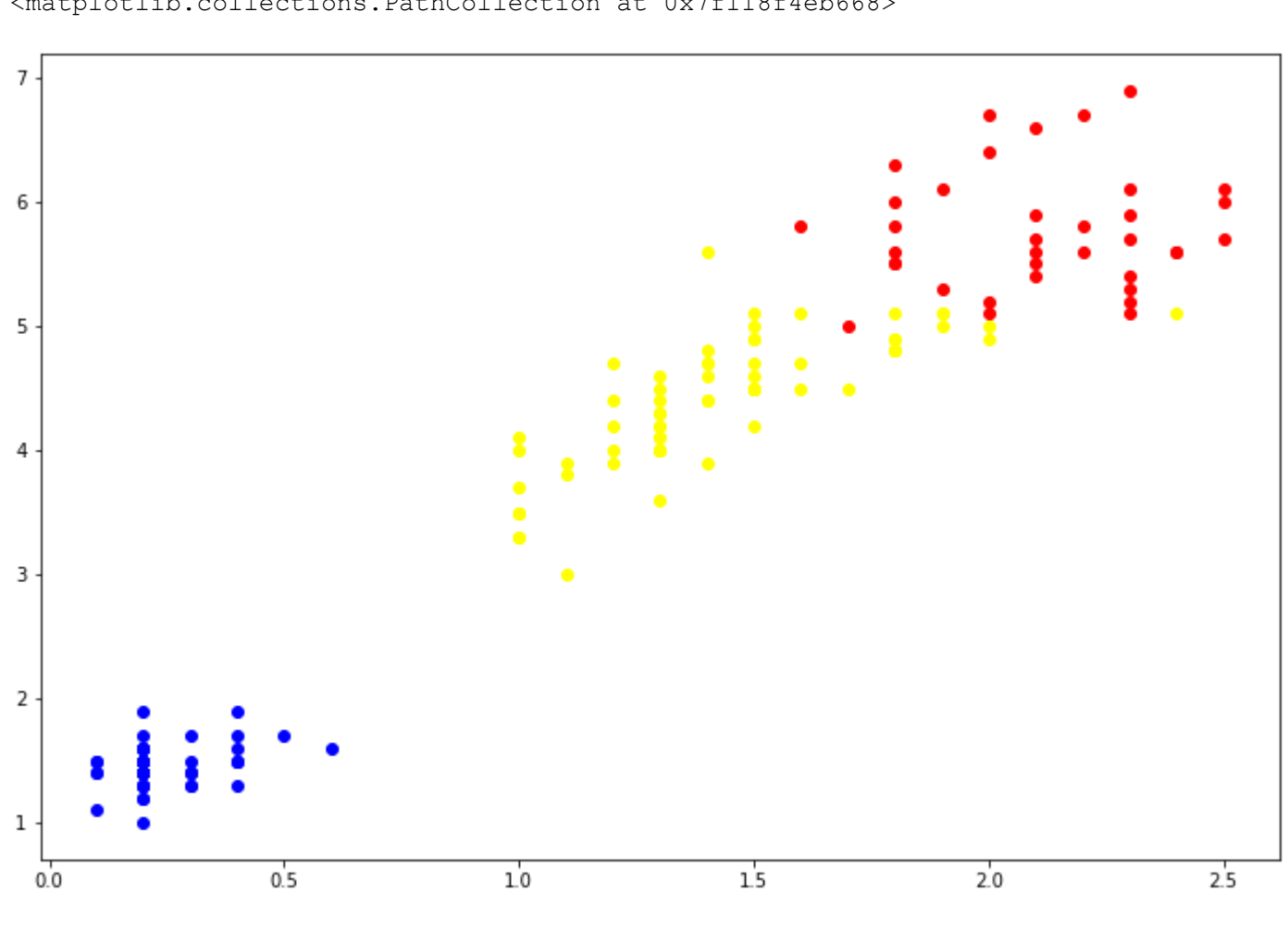
b) Train a clustering model using Bottom-up Agglomerative clustering.

Visualize your clusters on a 2-D plot. Choose any 2 dimensions from the 4 dimensions to plot. Try to pick the 2 dimensions that best separate the data. Your plot should contains all the data points with points from the same predicted cluster haveing the same color.

In [195]:
```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=3,affinity="euclidean",linkage="ward")
clustering.fit(X)
labels=clustering.labels_
```

In [196]:
```
plt.figure(figsize=(12,8))
plt.scatter(X[labels==0,3],X[labels==0,2],color='yellow')
plt.scatter(X[labels==1,3],X[labels==1,2],color='blue')
plt.scatter(X[labels==2,3],X[labels==2,2],color='red')
```

Out[196]: `<matplotlib.collections.PathCollection at 0x7f118f4eb668>`

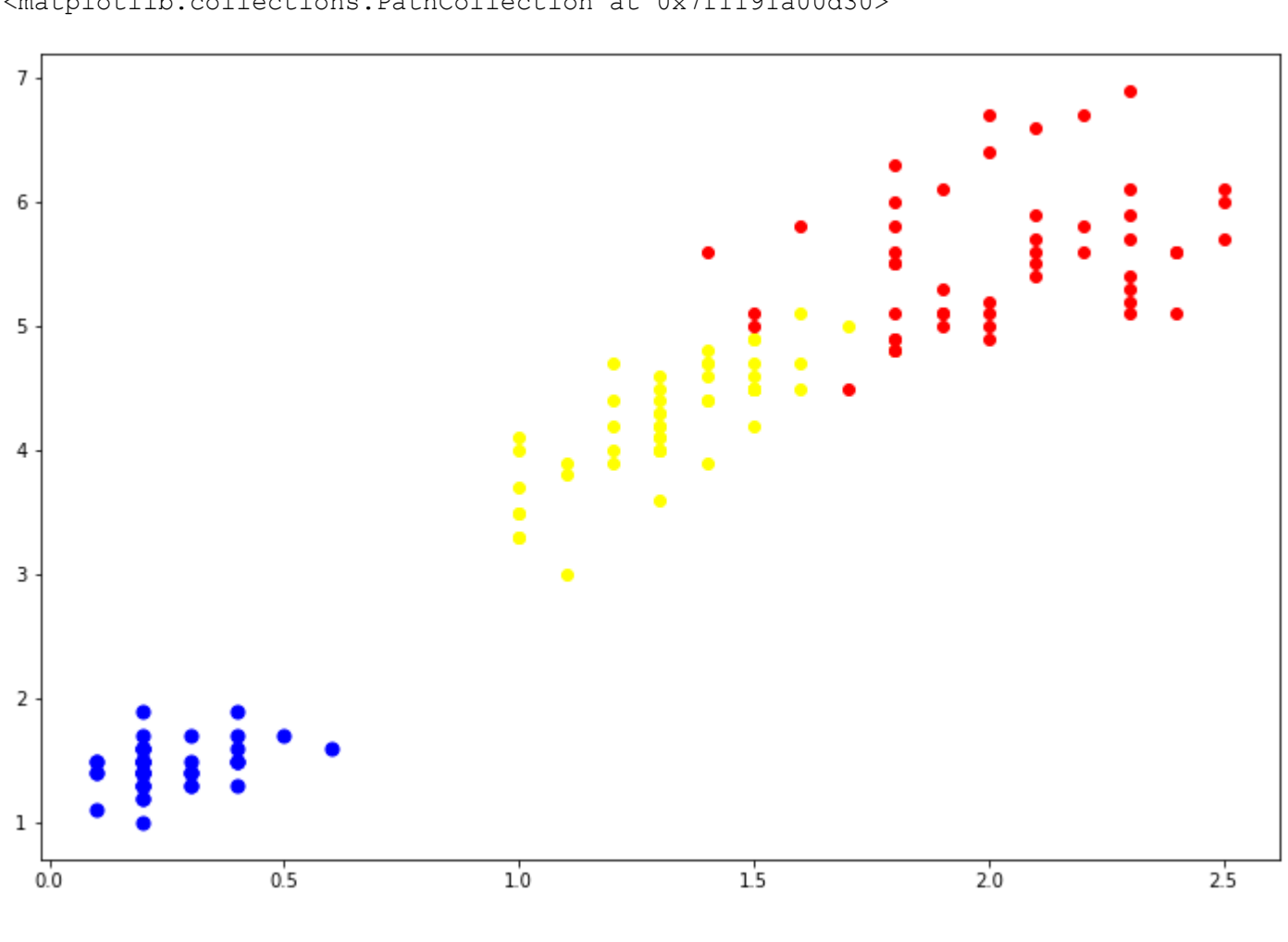


Repeat the visualization step above using the same 2 dimensions. This time, plot according to the ground truth classes. Comment on the performance of your model.

In [6]:
```
plt.figure(figsize=(12,8))
plt.scatter(data[data[4]=="Versicolor"][3],data[data[4]=="Versicolor"][2],color='yellow')
plt.scatter(data[data[4]=="Virginica"][3],data[data[4]=="Virginica"][2],color='red')
plt.scatter(data[data[4]=="Setosa"][3],data[data[4]=="Setosa"][2],s=50,color="blue")

# As two graphs almost align, the performance is good especailly for Setosa, Versicolor and Virginica h
ave some overlaps for three points.
```

Out[6]: `<matplotlib.collections.PathCollection at 0x7f1191a00d30>`



c) Train a clustering model using K-mean clustering.

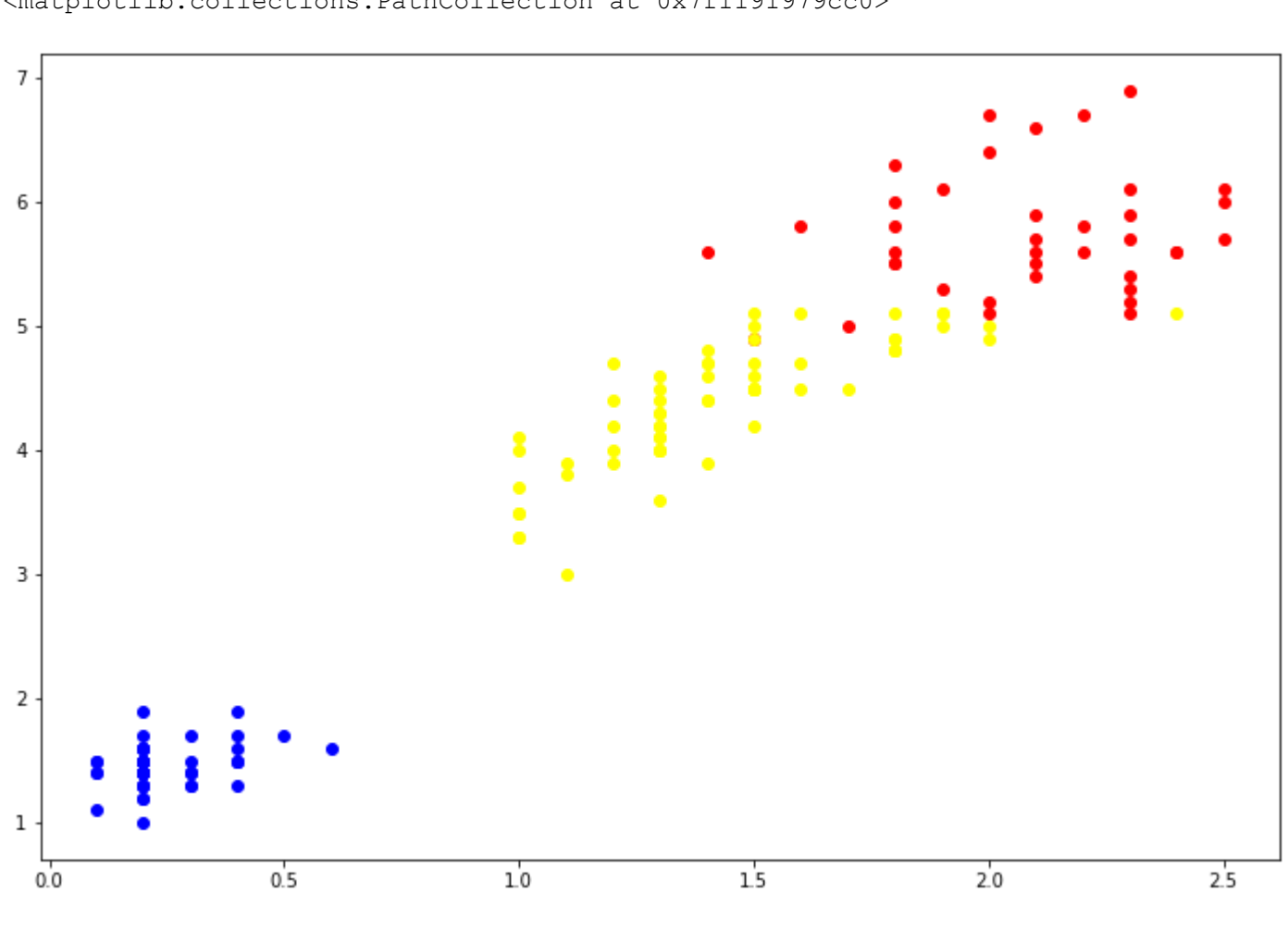
In [7]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
klabels = kmeans.labels_
```

Visualize your clusters on a 2-D plot. Choose any 2 dimensions from the 4 dimensions to plot. Try to pick the 2 dimensions that best separate the data. Your plot should contains all the data points with points from the same predicted cluster haveing the same color.

In [8]:
```
plt.figure(figsize=(12,8))
plt.scatter(X[klabels==0,3],X[klabels==0,2],color='red')
plt.scatter(X[klabels==1,3],X[klabels==1,2],color='blue')
plt.scatter(X[klabels==2,3],X[klabels==2,2],color='yellow')
```

Out[8]: `<matplotlib.collections.PathCollection at 0x7f1191979cc0>`

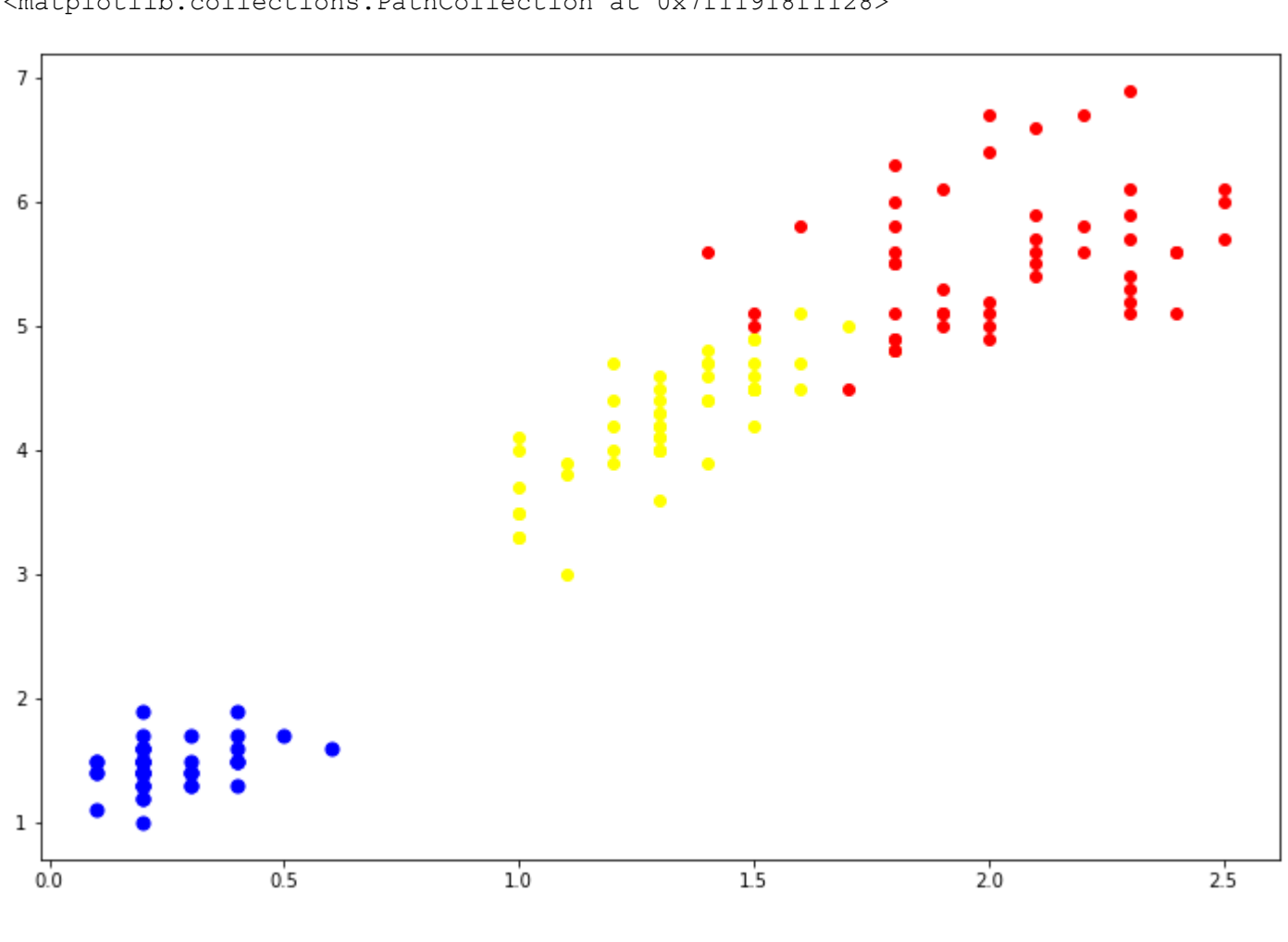


Repeat the visualization step above using the same 2 dimensions. This time, plot according to the ground truth classes. Comment on the performance of your model.

In [9]:
```
plt.figure(figsize=(12,8))
plt.scatter(data[data[4]=="Versicolor"][3],data[data[4]=="Versicolor"][2],color='yellow')
plt.scatter(data[data[4]=="Virginica"][3],data[data[4]=="Virginica"][2],color='red')
plt.scatter(data[data[4]=="Setosa"][3],data[data[4]=="Setosa"][2],s=50,color="blue")

# As two graphs almost align, the performance is good especailly for Setosa. Versicolor and Virginica h
ave some overlaps for three points.
```

Out[9]: `<matplotlib.collections.PathCollection at 0x7f11918f1128>`



d) Perform Principle Component Analysis (PCA) on the data. Project the original data on the 2 largest principle components. Store this new projected 2-D data in a variable ***X_projected***.

In [10]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2).fit(X)
X_pca = pca.transform(X)
X_pca
```

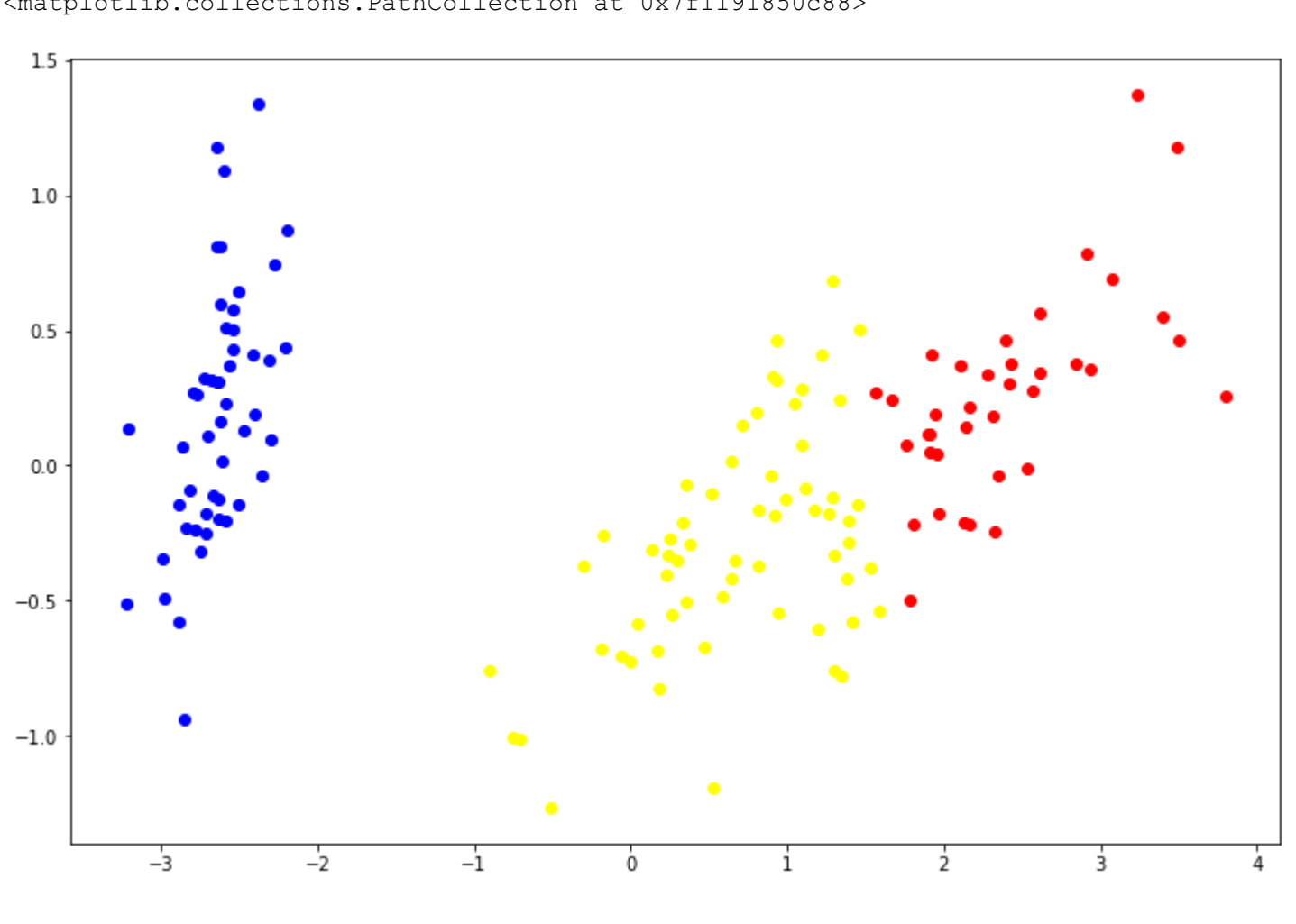
Out[10]:
```
array([[-2.68412563,  0.31939725],
       [-2.71414169, -0.17700123],
       [-2.88899057, -0.14494943],
       [-2.74534286, -0.31829898],
       [-2.72871654,  0.32675451],
       [-2.28085963,  0.74133045],
       [-2.82053775, -0.08946138],
       [-2.62614497,  0.16338496],
       [-2.88638273, -0.57831175],
       [-2.6727558 , -0.11377425],
       [-2.50694709,  0.6450689 ],
       [-2.61275523,  0.01472994],
       [-2.78610927, -0.235112  ],
       [-3.22380374, -0.51139459],
       [-2.64475039,  1.17876464],
       [-2.38603903,  1.33806233],
       [-2.62352788,  0.81067951],
       [-2.64829671,  0.31184914],
       [-2.19982032,  0.87283904],
       [-2.5879864 ,  0.51356031],
       [-2.31025622,  0.39134594],
       [-2.54370523,  0.43299606],
       [-3.21593942,  0.13346807],
       [-2.30273318,  0.09870885],
       [-2.35575405, -0.03728186],
       [-2.50666891, -0.14601688],
       [-2.46882007,  0.13095149],
       [-2.56231991,  0.36771886],
       [-2.63953472,  0.31203998],
       [-2.63198939, -0.19696122],
       [-2.58739848, -0.20431849],
       [-2.4099325 ,  0.41092426],
       [-2.64886233,  0.81336382],
       [-2.59873675,  1.09314576],
       [-2.63692688, -0.12132235],
       [-2.86624165,  0.06936447],
       [-2.62523805,  0.59937002],
       [-2.80068412,  0.26864374],
       [-2.98050204, -0.48795834],
       [-2.59000631,  0.22904384],
       [-2.77010243,  0.26352753],
       [-2.84936871, -0.94096057],
       [-2.99740655, -0.34192606],
       [-2.40561449,  0.18887143],
       [-2.20948924,  0.43666314],
       [-2.71445143, -0.2502082 ],
       [-2.53814826,  0.50377114],
       [-2.83946217, -0.22794557],
       [-2.54308575,  0.57941002],
       [-2.70335978,  0.10770608],
       [ 1.28482569,  0.68516047],
       [ 0.93248853,  0.31833364],
       [ 1.46430232,  0.50426282],
       [ 0.18331772, -0.82795901],
       [ 1.08810326,  0.07459068],
       [ 0.64166908, -0.41824687],
       [ 1.09506066,  0.28346827],
       [-0.74912267, -1.00489096],
       [ 1.04413183,  0.2283619 ],
       [-0.0087454 , -0.72308191],
       [-0.50784088, -1.26597119],
       [ 0.51169856, -0.10398124],
       [ 0.26497651, -0.55003646],
       [ 0.98493451, -0.12481785],
       [-0.17392537, -0.25485421],
       [ 0.92786078,  0.46717949],
       [ 0.66028376, -0.35296967],
       [ 0.23610499, -0.33361077],
       [ 0.94473373, -0.54314555],
       [ 0.04522698, -0.58383438],
       [ 1.11628318, -0.08461685],
       [ 0.35788842, -0.06892503],
       [ 1.29818388, -0.32778731],
       [ 0.92172892, -0.18273779],
       [ 0.71485333,  0.14905594],
       [ 0.90017437,  0.32850447],
       [ 1.33202444,  0.24444088],
       [ 1.55780216,  0.26749545],
       [ 0.81329065, -0.1633503 ],
       [-0.30558378, -0.36826219],
       [-0.06812649, -0.70517213],
       [-0.18962247, -0.68028676],
       [ 0.13642871, -0.31403244],
       [ 1.38002644, -0.42095429],
       [ 0.58800644, -0.48428742],
       [ 0.80685831,  0.19418231],
       [ 1.22069088,  0.40761959],
       [ 0.81509524, -0.37203706],
       [ 0.24595768, -0.2685244 ],
       [ 0.16641322, -0.68192672],
       [ 0.46480029, -0.67071154],
       [ 0.8908152 , -0.03446444],
       [ 0.23054802, -0.40438585],
       [-0.70453176, -1.01224823],
       [ 0.35698149, -0.50491009],
       [ 0.33193448, -0.21265468],
       [ 0.37621565, -0.29321893],
       [ 0.64257601,  0.01773819],
       [-0.90646986, -0.75609337],
       [ 0.29900084, -0.34889781],
       [ 2.53119273, -0.00984911],
       [ 1.41523588, -0.57491635],
       [ 2.61667602,  0.34390315],
       [ 1.97153105, -0.1797279 ],
       [ 2.35000592, -0.04026095],
       [ 3.39703874,  0.55083667],
       [ 0.52123224, -1.19275873],
       [ 2.93258707,  0.3555    ],
       [ 2.32122882, -0.2438315 ],
       [ 2.91675097,  0.78279195],
       [ 1.66177415,  0.24222841],
       [ 1.80340195, -0.21563762],
       [ 2.1655918 ,  0.21627559],
       [ 1.34616358, -0.77681835],
       [ 1.58592822, -0.53964071],
       [ 1.90445637,  0.11925069],
       [ 1.94968906,  0.04194326],
       [ 3.48705536,  1.17573933],
       [ 3.79564542,  0.25732297],
       [ 1.30079171, -0.76114964],
       [ 2.42781791,  0.37819601],
       [ 1.19900111, -0.60609153],
       [ 3.49992004,  0.4606741 ],
       [ 1.38876613, -0.20439933],
       [ 2.2754305 ,  0.33499061],
       [ 2.61409047,  0.56090136],
       [ 1.25850816, -0.17970479],
       [ 1.29113206, -0.11666865],
       [ 2.12360872, -0.20972948],
       [ 2.38800302,  0.4646398 ],
       [ 2.84167278,  0.37526917],
       [ 3.23067366,  1.37416509],
       [ 2.15943764, -0.21727758],
       [ 1.44416124, -0.14341341],
       [ 1.78129481, -0.49990168],
       [ 3.07649993,  0.68808568],
       [ 2.14424331,  0.1400642 ],
       [ 1.90509815,  0.04930053],
       [ 1.16932634, -0.16499026],
       [ 2.10761114,  0.37228787],
       [ 2.31415471,  0.18365128],
       [ 1.9222678 ,  0.40920347],
       [ 1.41523588, -0.57491635],
       [ 2.56301338,  0.2778626 ],
       [ 2.41874618,  0.3047982 ],
       [ 1.94410979,  0.1875323 ],
       [ 1.52716661, -0.37531698],
       [ 1.76434572,  0.07885885],
       [ 1.90094161,  0.11662796],
       [ 1.39018886, -0.28266094]])
```

Repeat part **b** on the new 2-D data. Train the Bottom-up Agglomerative model and visualize your results.

In [11]:
```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=3,affinity="euclidean",linkage="ward")
clustering.fit(X_pca)
labels_pca=clustering.labels_

plt.figure(figsize=(12,8))
plt.scatter(X_pca[labels_pca==0,0],X_pca[labels_pca==0,1],color='yellow')
plt.scatter(X_pca[labels_pca==1,0],X_pca[labels_pca==1,1],color='blue')
plt.scatter(X_pca[labels_pca==2,0],X_pca[labels_pca==2,1],color='red')
```

Out[11]: `<matplotlib.collections.PathCollection at 0x7f1191850c88>`

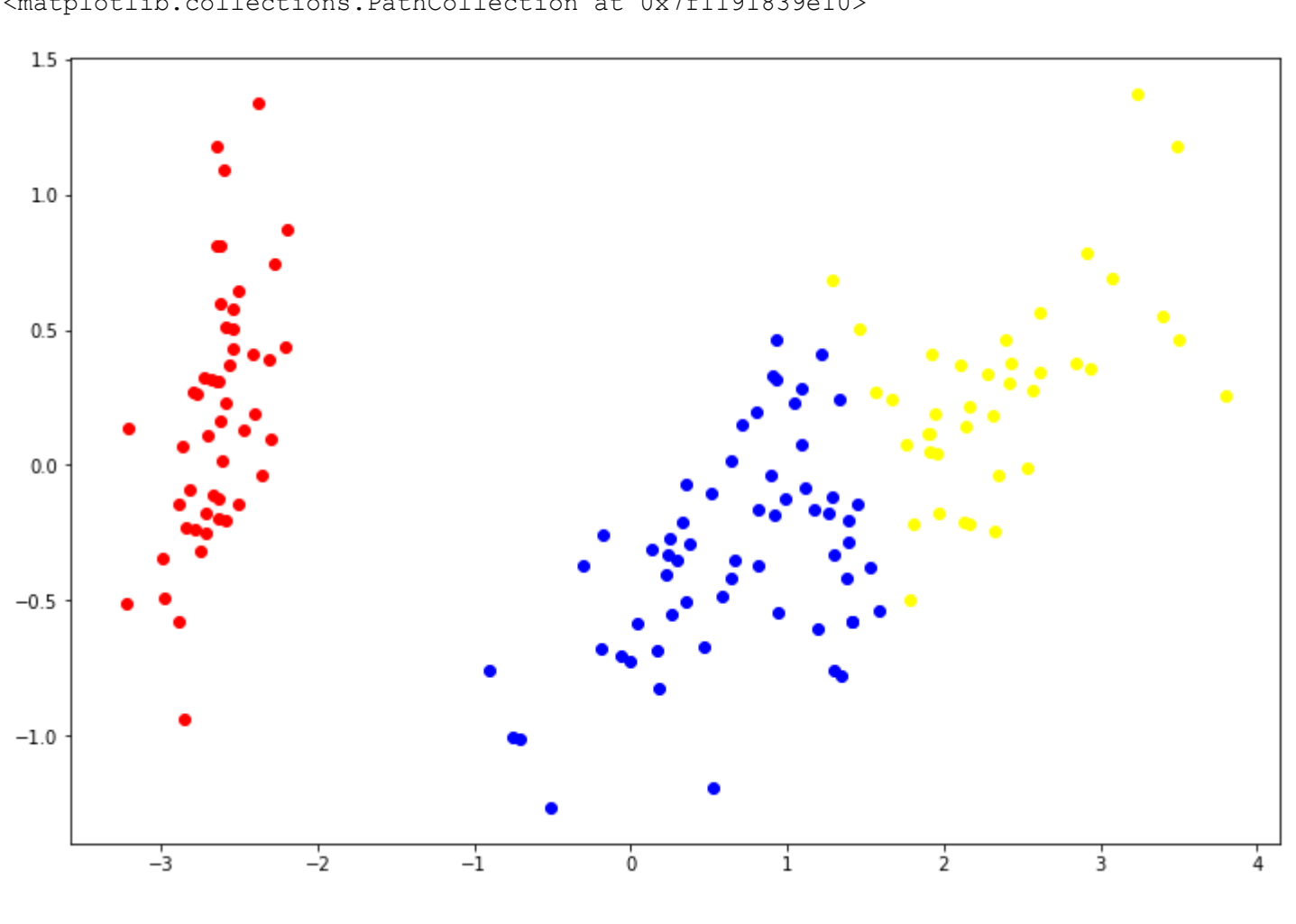


Repeat part **c** on the new 2-D data. Train the K-means model and visualize your result.

In [12]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=0).fit(X_pca)
klabels_pca = kmeans.labels_
plt.figure(figsize=(12,8))
plt.scatter(X_pca[klabels_pca==0,0],X_pca[klabels_pca==0,1],color='red')
plt.scatter(X_pca[klabels_pca==1,0],X_pca[klabels_pca==1,1],color='blue')
plt.scatter(X_pca[klabels_pca==2,0],X_pca[klabels_pca==2,1],color='yellow')
```

Out[12]: `<matplotlib.collections.PathCollection at 0x7f1191839e10>`



Compare the quality of 4-D and 2-D clusterings. When would the ideas of projection and dimensionality reduction be useful?

PCA used in combination with K means clustering dimension reduction was very useful and increase accuracy, but when PCA used with Bottom-up Agglomerative clustering doesn't change much.

Overall, 2-D has better partision than 4-D. Thus PCA will be slightly better. The higher dimension, the more useful PCA will be as better clustering would be formed.

Problem 2

In this problem, you will first implement the Naive Bayes (NB) algorithm from scratch. We will use a dataset for classifying if a patient has breast cancer. Each instance (row) in the dataset is a patient described by the 9 following features (with their corresponding values).

1. Clump Thickness: 1 - 10
2. Uniformity of Cell Size: 1 - 10
3. Uniformity of Cell Shape: 1 - 10
4. Marginal Adhesion: 1 - 10
5. Single Epithelial Cell Size: 1 - 10
6. Bare Nuclei: 1 - 10
7. Bland Chromatin: 1 - 10
8. Normal Nucleoli: 1 - 10
9. Mitoses: 1 - 10

Given these features, we will classify a car into one of the 2 classes: 0 (benign) or 1 (malignant).

NB is a very simple algorithm. Consider a feature **X**. For each value x_i of **X** and each class label y_j, NB calculates the value of $P(X = x_i | Y = y_j)$. For example, take the feature **Mitoses**, NB will calculate all the following values:

- P(Mitoses = 1 | Class = 0), P(Mitoses = 2 | Class = 0), P(Mitoses = 3 | Class = 0), ... , P(Mitoses = 10 | Class = 0)
- P(Mitoses = 1 | Class = 1), P(Mitoses = 2 | Class = 1), P(Mitoses = 3 | Class = 1), ... , P(Mitoses = 10 | Class = 1)

Repeat this calculation for all the features. In the end, NB keep a recording of all possible $P(X|Y)$. The calculation itself is intuitive:

$$P(X = x_i | Y = y_j) = \frac{\text{Number of rows with } X = x_i \text{ and } Y = y_j}{\text{Number of rows with } Y = y_j}$$

In addition, NB also calculate the priors probability $P(Y = y_j)$. Again, intuitively:

$$P(Y = y_i) = \frac{\text{Number of rows with } Y = y_j}{\text{Number of rows in the dataset}}$$

Given a test example $X_{test} = \{X_0 = x_0, X_1 = x_1, \ldots, X_i = x_i\}$, for each class label y_j, NB calculate:

$$\begin{aligned} P(Y = y_j | X_{test}) &= P(X_{test} | Y = y_j) P(Y = y_j) \\ &= P(X_0 = x_0 | Y = y_j) P(X_1 = x_1 | Y = y_j) \ldots P(X_i = x_i | Y = y_j) P(Y = y_j) \end{aligned}$$

Such calculation is easy since we have bookkept all $P(X|Y)$ and all $P(Y)$ in previous steps. The output of the model is simply:

$$\underset{y_j}{\operatorname{argmax}} P(Y = y_j | X_{test})$$

You will do each of these steps following this problem. We will use Pandas to deal with the data in this problem. Pandas can do queries like "Get all the rows in which Clump Thickness = 2 and Class = 1" with minimal syntax.

a) First let's load the dataset and store it in a Pandas dataframe. Play with the dataframe and get used to the queries (this part is not graded). This guide is a good place to start:

https://medium.com/python-in-plain-english/filtering-rows-and-columns-in-pandas-python-techniques-you-must-know-6cdfc32c614c

Split the dataset into a training set and a testing set. Use 10% of the data as the testing set (The splitting is graded).

In [211]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_csv("breast-cancer-wisconsin.csv")
X = data.iloc[:,0:9]
y = data.iloc[:,9]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=42)
```

b) Now we will build up the bookkeeping. Let's start with the priors $P(Y = y_j)$, $y_j \in$ {0,1}. For each of these labels, calculate $P(Y = y_j)$ by simply counting the number of times y_j appears in the dataset divided by the size of the dataset. You can bookkeep the priors in a dictionary with keys being y_j.

In [228]:
```
ones_percent = y_train[y_train == 1].size / y_train.size
zeros_percent = y_train[y_train == 0].size / y_train.size

dic = {"y1":ones_percent,
      "y0":zeros_percent}
dic["y1"]
```

Out[228]: `0.3449920508744038`

Now let's calculate the conditional probability $P(X = x_i | Y = y_j)$. If you do the query with Pandas, this counting should be simple. This time, store the conditional probability in another dictionary with keys (x_i, y_j) or any data structure that you prefer.

In [229]:
```
import numpy as np
ones= []
zeros = []
d1 = pd.DataFrame({"one":[]})
d2 = pd.DataFrame({"zero":[]})
for i in X.columns:
    for j in range (1,11):
        x1 = X_train[(X_train[i] == j) & (y_train ==1)]
        x0 = X_train[(X_train[i] == j) & (y_train ==0)]
        v1 = x1[i].size / y_train[y_train == 1].size
        v0 = x0[i].size / y_train[y_train == 0].size
        ones.append(v1)
        zeros.append(v0)
ones_m = pd.DataFrame(np.resize(ones, [9,10]))
ones_m.columns = [1,2,3,4,5,6,7,8,9,10]
ones_m.index = ["Clump Thickness", " Uniformity of Cell Size", " Uniformity of Cell Shape", " Marginal
 Adhesion", " Single Epithelial Cell Size", " Bare Nuclei", " Bland Chromatin", " Normal Nucleoli", "
 Mitoses"]
print(ones_m)
zeros_m = pd.DataFrame(np.resize(zeros, [9,10]))
zeros_m.columns = [1,2,3,4,5,6,7,8,9,10]
zeros_m.index = ["Clump Thickness", " Uniformity of Cell Size", " Uniformity of Cell Shape", " Margina
l Adhesion", " Single Epithelial Cell Size", " Bare Nuclei", " Bland Chromatin", " Normal Nucleoli", "
 Mitoses"]
print(zeros_m)
```

```
                                    1         2  ...         9        10
Clump Thickness              0.013825  0.018433  ...  0.055300  0.304147
 Uniformity of Cell Size     0.018433  0.036866  ...  0.023041  0.262673
 Uniformity of Cell Shape    0.009217  0.032258  ...  0.032258  0.239631
 Marginal Adhesion           0.129032  0.092166  ...  0.018433  0.216590
 Single Epithelial Cell Size 0.004608  0.101382  ...  0.009217  0.124424
 Bare Nuclei                 0.069124  0.036866  ...  0.036866  0.557604
 Bland Chromatin             0.009217  0.032258  ...  0.050691  0.082949
 Normal Nucleoli             0.175115  0.023041  ...  0.064516  0.248848
 Mitoses                     0.557604  0.119816  ...  0.000000  0.055300

[9 rows x 10 columns]
                                    1         2  ...         9        10
Clump Thickness              0.310680  0.104369  ...  0.000000  0.000000
 Uniformity of Cell Size     0.827670  0.082524  ...  0.002427  0.000000
 Uniformity of Cell Shape    0.762136  0.116505  ...  0.000000  0.000000
 Marginal Adhesion           0.817961  0.077670  ...  0.002427  0.002427
 Single Epithelial Cell Size 0.097087  0.796117  ...  0.000000  0.002427
 Bare Nuclei                 0.868932  0.048544  ...  0.000000  0.007282
 Bland Chromatin             0.332524  0.351942  ...  0.000000  0.000000
 Normal Nucleoli             0.885922  0.065534  ...  0.002427  0.000000
 Mitoses                     0.973301  0.014563  ...  0.000000  0.000000

[9 rows x 10 columns]
```

That's it! You have successfully "trained" a NB model. Let's test our model on the test dataset.

Implement a **predict** function that returns $\underset{y_j}{\operatorname{argmax}} P(Y = y_j | X_{test})$ (refer to the description above).

In [230]:
```
results = []
X_test= X_test.reset_index(drop=True)
for i in range (0, len(y_test)):
    rones = 1
    rzeros = 1
    for j in X_test.columns:
        v = X_test.at[i,j]
        rones *= ones_m.at[j,v]
        rzeros *= zeros_m.at[j,v]
    rones *=  ones_percent
    rzeros *=  zeros_percent
    if (rones < rzeros):
        results.append(0)
    else:
        results.append(1)
```

Predict the label of all the instances in the test dataset, calculate and print out the accuracy.

In [231]:
```
accuracy_score(y_test, results)
```

Out[231]: `0.9857142857142858`

Problem 3

In this problem, you will implement the Logistic Regression (LR) algorithm from scratch. Similar to NB, LR relies on $P(Y|X)$ to predict the class of an example. However, unlike NB, a generative model, LR is a discriminative model so it does not need to estimate $P(X|Y)$ and $P(Y)$. LR assumes the form of the conditional probability $P(Y|X)$ to be:

$$P(Y|X) = f(X) = \frac{1}{1 + e^{-(\alpha X + \beta)}}$$

$f(X)$ returns a value in (0,1). The model classifies X as 1 if $f(X)$ is closer to 1 and 0 otherwise. We have to estimate the model parameters: the vector α and β from the data, which we will do via stochastic gradient descent (SGD). In SGD, a training example is shown to the model each at a time. The model makes a prediction on the training example and the error between the prediction and the ground-truth label is used to update the model's parameters. We use the log-likelihood loss to estimate the error in this problem. In particular, the log-likelihood loss for classifying the $X^{(i)}$ example with the ground-truth $y^{(i)}$ is:

$$LL(y^{(i)}, f(X^{(i)})) = -(y^{(i)} \log(f(X^{(i)})) + (1 - y^{(i)}) \log(1 - f(X^{(i)})))$$

In our case, the updating is as the followings:

$$\alpha_j(t+1) = \alpha_j(t) - \frac{\partial LL}{\partial \alpha_j} \times rate$$
$$\beta(t+1) = \beta(t) - \frac{\partial LL}{\partial \beta} \times rate$$

where $rate$ is the how much change we want to make to the parameters in each update. Whenever we finish looping through all the instances in the dataset to update the parameters, we finish a training epoch. We may do many training epochs (looping through the dataset many times), until the model parameters converge.

First, we need to derive the formula for $\frac{\partial LL}{\partial \alpha_j}$ and $\frac{\partial LL}{\partial \beta}$. To keep the problem simple, these are provided to you:

$$\frac{\partial LL}{\partial \alpha_j} = -(y^{(i)} - f(X^{(i)})) X_j^{(i)}$$
$$\frac{\partial LL}{\partial \beta} = -(y^{(i)} - f(X^{(i)}))$$

a) We use the same dataset for predicting breast cancer in Problem 2 for this problem. Load the dataset and split it into a training set and a testing set.

In [88]:
```
from sklearn.model_selection import train_test_split

data = pd.read_csv("breast-cancer-wisconsin.csv")
X = data.iloc[:,0:9]
y = data.iloc[:,9]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

b) Learn the model parameters using SGD. A skeleton for SGD is provided to help you understand the process (You don't have to use the provided code).

In [89]:
```
from math import exp

epochs = 20
rate = 0.01
SGD = pd.concat([X_train,y_train], axis=1).to_numpy()
coefficient = np.zeros(len(SGD[0]))

def predict(row, coefficient):
    y_ = coefficient[0]  # beta
    for i in range(1,len(row)): # alpha for corresponding to each x
      y_ += coefficient[i]*row[i-1]
    return 1.0/(1.0+exp(-y_))

for epoch in range(epochs):
  all_error = 0
  for row in SGD:
    yi = predict(row,coefficient)
    error = row[-1] - yi
    all_error += error**2
    coefficient[0] += -rate*error*yi*(yi-1)
    # update parameters
    for i in range(1,len(row)):
      coefficient[i] += -rate*error*yi*(yi-1)*row[i-1]

print(coefficient)
```

```
[-1.84926149 -0.10111436  0.59965477  0.18992236  0.02682314 -0.34644695
  0.40399392 -0.27003814  0.29796996 -0.18779661]
```

c) Load the testing dataset. Use $f(X)$ to predict the labels of the test instances. Calculate the accuracy.

In [189]:
```
y_result = []
test = X_test.to_numpy()
for r in test:
  yi = round(predict(r,coefficient))
  y_result.append(yi)
accuracy_score(y_test,y_result)
```

Out[189]: `0.9714285714285714`

Problem 4

In this problem, you will investigate how the size of the training data on the performance of a generative model (NB) and a discriminative model (LR). **You are free to use scikit-learn's NB and LR.**

a) Load the breast cancer training dataset and split it into a training set and a testing set.

In [62]:
```
data = pd.read_csv("breast-cancer-wisconsin.csv")
X = data.iloc[:,0:9]
y = data.iloc[:,9]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.10)
```

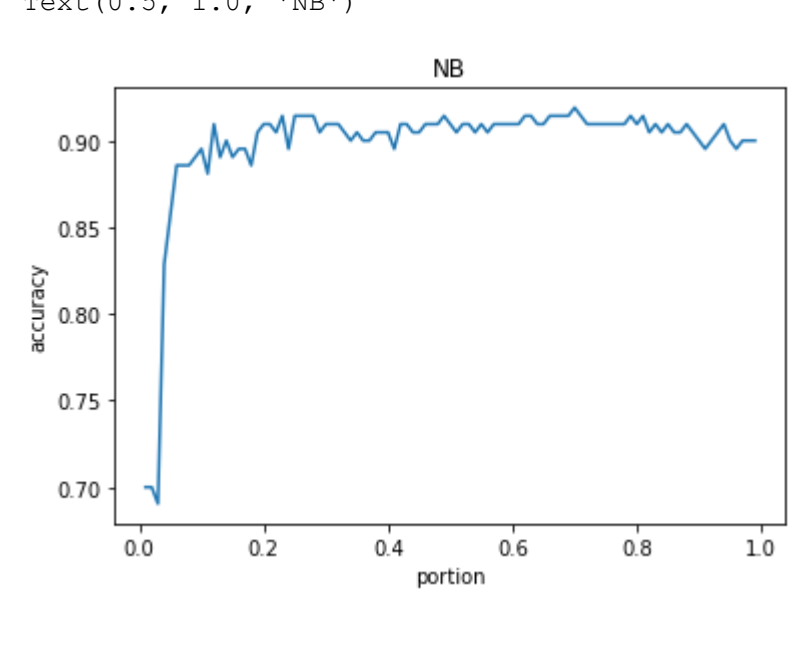
b) We will train NB classifiers that use only a portion of the training data. Let's vary this portion from 1% to 100% of the training set (100 iterations). For each portion value, sample the training set according to the portion, train a NB classifer on this portion of the training set, and evaluate the model accuracy on the testing set.

In [116]:
```
from sklearn.naive_bayes import MultinomialNB
sample=[]
accuracy=[]
for i in range(1,100):
  if i is not 100:
    X1,X0,y1,y0=train_test_split(X_train,y_train,test_size = 1-i/100.0,random_state=10)
  else:
    X1 = X_train
    y1 = y_train
  clf = MultinomialNB()
  clf.fit(X1,y1)
  X_ = clf.predict(X_test)
  sample.append(i/100.0)
  accuracy.append(accuracy_score(y_test,X_))
```

Plot the accuracies of the models versus the portion of the training set used. Comment on the effect of increasing the size of the training dataset on the performance of NB.

In [117]:
```
plt.plot(sample,accuracy)
plt.xlabel('portion')
plt.ylabel('accuracy')
plt.title("NB")
```

Out[117]: `Text(0.5, 1.0, 'NB')`



The larger the training dataset, the better the prediction espeically before 20% portion of dataset. Afterwards, the accuracy is about the same.

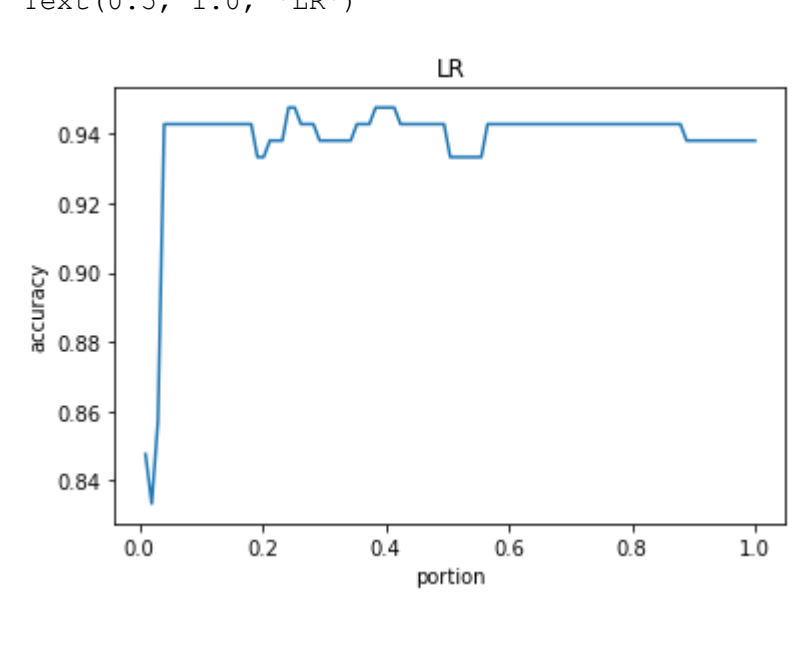
c) Repeat part **b** with LR.

In [126]:
```
from sklearn.linear_model import LogisticRegression

sample=[]
accuracy=[]
for i in range(1,100):
  if i is not 100:
    X1,X0,y1,y0=train_test_split(X_train,y_train,test_size = 1-i/100.0,random_state=10)
  else:
    X1 = X_train
    y1 = y_train
  clf = LogisticRegression()
  clf.fit(X1,y1)
  X_ = clf.predict(X_test)
  sample.append(i/100.0)
  accuracy.append(accuracy_score(y_test,X_))

plt.plot(sample,accuracy)
plt.xlabel('portion')
plt.ylabel('accuracy')
plt.title("LR")
```

Out[126]: `Text(0.5, 1.0, 'LR')`



Between a generative model (NB) and a discriminative model (LR), which is more affected by small training dataset? Think about a reason for this.

NB is more affected by small training dataset, as the accuracy of small training dataset is better for LR (As the graphs shown, at the beginning of the graph, the accuracy of LR vs NR is 85% vs 70%). Native Bayes models makes it assumes features in the datasets are independent thus is more affected by a small training dataset. The discriminative model learned boundaries between features/classes thus might not need many training datasets to begin to learn. That is to say, as Generative model depends on the joint probability it works well with smaller dataset , but discriminative models tend to perform poor with fewer dataset as it will overfit.

Problem 5

What is the hyperparameters, the parameters, and the objective function for finding the best hypothesis for each of the following methods:

1. Decision Tree (Entropy gain)
2. Support Vector Machine
3. K-nearest Neighbors Clustering

1.**Hyperparameters**: depth of the tree, purity, splitter, estimator, maximum leaf nodes, depth of the trees, minimumimpurity decrease and impurity split, minimum number of sample leafs, number of features.

Parameter: nodes, attribute threshold

Objective function: The Entropy Gain: maximize the information gain at each split or optimizedduring the fitting.
$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_n|}{|S|} Entropy(S_v)$

2.**Hyperparameters:** kernel, polynomial degree, C, gamma, margin, shrinking heuristic (bool)

Parameters: maximum margin of the support vectors and the w,b variables

Objective function: To find a plane that has the maximum margin, or 'optimizing' in theconstraints. kernel function:
$g(x) = b + w^T \phi(x) = b + \sum_{i \in SV} \alpha_i y_i K(x, x_i) > 0$

3.**Hyperparameters:** the number of clusters, complexity, weighted neighbors, precomputedistances, algorithm to use (such as ball_tree or brute), Leaf size passed to BallTree or KDTree,Power parameter for the Minkowski metric, metric.

Parameters: the number of centroid, measure of distance, decision boundary

Objective function: the function we used to calculate and optimize the distance such as Euclidean,Minkowski or Manhattan.
$\hat{t} = argmax_c \sum_{i: x_i \in N_k(x, \hat{x})} \delta(t_i, c)$