Question 1

In [114]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split as split_data
from sklearn.preprocessing import StandardScaler
```

In [115]:

```
dataset = pd.read_csv("Iris.csv", header=0)
dataset.head()
```

Out[115]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [116]:

```
def convert(vector):
    visited = []
    ind = 0
    result = []
    for i in vector:
        if i in visited:
            result.append(visited.index(i))
        else:
            visited.append(i)
            result.append(len(visited)-1)
    return result
```

In [117]:

```
dataset.iloc[:, -1] = np.array(convert(dataset.iloc[:, -1].tolist()))
```

In [118]:

```
x = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
x.drop('Id', axis=1, inplace=True)
x.shape
```

Out[118]:

```
(150, 4)
```

In [119]:

```
scaler = StandardScaler()
```

In [120]:

```
x = scaler.fit_transform(x)
```

In [121]:

```
def covariance_matrix(x):
    return np.dot(x.T, x)/x.shape[0]
```

In [122]:

```
c = covariance_matrix(x)
c
```

Out[122]:

```
array([[ 1.        , -0.10936925,  0.87175416,  0.81795363],
       [-0.10936925,  1.        , -0.4205161 , -0.35654409],
       [ 0.87175416, -0.4205161 ,  1.        ,  0.9627571 ],
       [ 0.81795363, -0.35654409,  0.9627571 ,  1.        ]])
```

In [123]:

```
#finding eigen-values and eigen-vectors for covariance matrix
def eigen_values_vectors(x):
    eigenValues, eigenVectors = np.linalg.eig(x)
    # sort eigenvalues descending and select columns based on n_components
    n_cols = np.flip(np.argsort(eigenValues))
    selected_eigen_vectors = eigenVectors[:, n_cols]
    return np.flip(np.sort(eigenValues)),selected_eigen_vectors
```

In [124]:

```
eigenValues, eigenVectors = eigen_values_vectors(c)
print(eigenValues)
print("\n")
print(eigenVectors)
```

```
[2.91081808 0.92122093 0.14735328 0.02060771]


[[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]
```

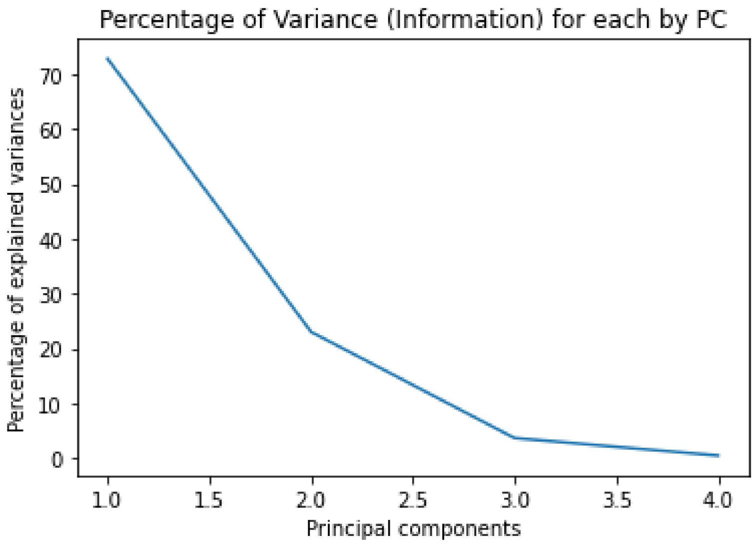
In [125]:

```
#Plot the principal components and percentage of explained variances.

explained_variances = [i/np.sum(eigenValues) for i in eigenValues]

print("Sum of  Explained Variance :",np.sum(explained_variances))
print("Explained Variance :", explained_variances)
```

```
Sum of  Explained Variance : 1.0
Explained Variance : [0.7277045209380137, 0.23030523267680617, 0.03683831957
627382, 0.005151926808906351]
```

In [126]:

```
#Plot the principal components and percentage of explained variances.
import matplotlib.pyplot as plt
X_axis = [1,2,3,4]
q = [i*100 for i in explained_variances]

plt.plot(X_axis,q)
plt.xlabel('Principal components')
plt.ylabel('Percentage of explained variances')
plt.title('Percentage of Variance (Information) for each by PC')
plt.show()
```



In [127]:

```
#consider only the features upto where the pov adds to <95% here 2
k = 2
eign = eigenVectors[:,:k]
eign
```

Out[127]:

```
array([[ 0.52237162, -0.37231836],
       [-0.26335492, -0.92555649],
       [ 0.58125401, -0.02109478],
       [ 0.56561105, -0.06541577]])
```

In [128]:

```
#transform into original matrix
X_transformed = np.dot(eign.T,x.T).T
print("Transformed X :\n", np.around(X_transformed, 2))
```

```
 [ 2.32 -2.63]
 [ 1.86  0.18]
 [ 1.11  0.3 ]
 [ 1.2   0.82]
 [ 2.8  -0.84]
 [ 1.58 -1.07]
 [ 1.35 -0.42]
 [ 0.92 -0.02]
 [ 1.85 -0.67]
 [ 2.02 -0.61]
 [ 1.9  -0.69]
 [ 1.15  0.7 ]
 [ 2.04 -0.86]
 [ 2.   -1.05]
 [ 1.87 -0.38]
 [ 1.56  0.91]
 [ 1.52 -0.27]
 [ 1.38 -1.02]
 [ 0.96  0.02]]
```

Question 2

In [129]:

```
data = pd.read_csv("framingham.csv", header=0)
data.head()
```

Out[129]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp
0	1	39	4.0	0	0.0	0.0	0	0
1	0	46	2.0	0	0.0	0.0	0	0
2	1	48	1.0	1	20.0	0.0	0	0
3	0	61	3.0	1	30.0	0.0	0	1
4	0	46	3.0	1	23.0	0.0	0	0

In [130]:

```
data = data.dropna() #drop all columns that contains na
```

In [131]:

```
X = data.iloc[:, :-1]
Y = data.iloc[:, -1]
```

In [132]:

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [137]:

```
c2 = np.round(covariance_matrix(X), 2)
c2
```

Out[137]:

```
array([[ 1.  , -0.02,  0.02,  0.21,  0.33, -0.05, -0.  ,  0.  ,  0.01,
        -0.07, -0.05,  0.05,  0.07, -0.11,  0.  ],
       [-0.02,  1.  , -0.16, -0.21, -0.19,  0.13,  0.05,  0.31,  0.11,
         0.27,  0.39,  0.21,  0.14, -0.  ,  0.12],
       [ 0.02, -0.16,  1.  ,  0.03,  0.01, -0.01, -0.03, -0.08, -0.04,
        -0.01, -0.12, -0.06, -0.14, -0.06, -0.03],
       [ 0.21, -0.21,  0.03,  1.  ,  0.77, -0.05, -0.04, -0.11, -0.04,
        -0.05, -0.13, -0.12, -0.16,  0.05, -0.05],
       [ 0.33, -0.19,  0.01,  0.77,  1.  , -0.05, -0.04, -0.07, -0.04,
        -0.03, -0.09, -0.06, -0.09,  0.06, -0.05],
       [-0.05,  0.13, -0.01, -0.05, -0.05,  1.  ,  0.11,  0.26,  0.05,
         0.09,  0.27,  0.2 ,  0.11,  0.01,  0.05],
       [-0.  ,  0.05, -0.03, -0.04, -0.04,  0.11,  1.  ,  0.07,  0.01,
         0.01,  0.06,  0.06,  0.04, -0.02,  0.02],
       [ 0.  ,  0.31, -0.08, -0.11, -0.07,  0.26,  0.07,  1.  ,  0.08,
         0.17,  0.7 ,  0.62,  0.3 ,  0.15,  0.09],
       [ 0.01,  0.11, -0.04, -0.04, -0.04,  0.05,  0.01,  0.08,  1.  ,
         0.05,  0.1 ,  0.05,  0.09,  0.06,  0.61],
       [-0.07,  0.27, -0.01, -0.05, -0.03,  0.09,  0.01,  0.17,  0.05,
         1.  ,  0.22,  0.17,  0.12,  0.09,  0.05],
       [-0.05,  0.39, -0.12, -0.13, -0.09,  0.27,  0.06,  0.7 ,  0.1 ,
         0.22,  1.  ,  0.79,  0.33,  0.18,  0.13],
       [ 0.05,  0.21, -0.06, -0.12, -0.06,  0.2 ,  0.06,  0.62,  0.05,
         0.17,  0.79,  1.  ,  0.39,  0.18,  0.06],
       [ 0.07,  0.14, -0.14, -0.16, -0.09,  0.11,  0.04,  0.3 ,  0.09,
         0.12,  0.33,  0.39,  1.  ,  0.07,  0.08],
       [-0.11, -0.  , -0.06,  0.05,  0.06,  0.01, -0.02,  0.15,  0.06,
         0.09,  0.18,  0.18,  0.07,  1.  ,  0.1 ],
       [ 0.  ,  0.12, -0.03, -0.05, -0.05,  0.05,  0.02,  0.09,  0.61,
         0.05,  0.13,  0.06,  0.08,  0.1 ,  1.  ]])
```

In [139]:

```
eigenValues2, eigenVectors2 = eigen_values_vectors(c2)
eigenValues2 = np.round(eigenValues2, 2)
eigenVectors2 = np.round(eigenVectors2, 2)
print(eigenValues2)
print("\n")
print(eigenVectors2)
```

```
[3.23 1.88 1.57 1.12 1.06 1.04 1.01 0.87 0.79 0.7  0.58 0.39 0.38 0.22
 0.17]


[[ 0.06  0.37  0.04  0.51 -0.22  0.12 -0.23 -0.19  0.01 -0.59  0.28  0.01
   0.02  0.12  0.08]
 [-0.3  -0.09  0.03  0.12 -0.22 -0.49 -0.17  0.06  0.34 -0.26 -0.59 -0.02
  -0.11 -0.01 -0.15]
 [ 0.11 -0.01 -0.03 -0.03  0.64  0.22 -0.58 -0.3   0.04 -0.05 -0.31  0.01
  -0.01 -0.01  0.03]
 [ 0.2   0.59  0.05 -0.1   0.07 -0.16  0.06  0.1   0.03  0.23 -0.19 -0.01
  -0.01  0.68 -0.03]
 [ 0.17  0.63  0.04 -0.03  0.   -0.13  0.02  0.03 -0.01  0.12 -0.13 -0.01
  -0.01 -0.72 -0.01]
 [-0.21  0.04 -0.05  0.16  0.49 -0.18  0.13  0.57 -0.45 -0.32  0.02 -0.03
  -0.06 -0.   -0.04]
 [-0.07 -0.02 -0.02  0.38  0.38 -0.23  0.59 -0.53  0.13  0.06 -0.01  0.01
  -0.01 -0.    0.01]
 [-0.43  0.16 -0.12  0.    0.09  0.1  -0.04  0.09  0.22  0.08  0.08  0.35
   0.74 -0.   -0.14]
 [-0.13 -0.01  0.68  0.06  0.04  0.05 -0.03  0.03 -0.01  0.1   0.05  0.64
  -0.3  -0.01  0.01]
 [-0.19  0.02 -0.   -0.23 -0.02 -0.62 -0.34 -0.35 -0.36  0.1   0.38  0.01
   0.05  0.02  0.  ]
 [-0.48  0.15 -0.1  -0.05  0.04  0.07 -0.04  0.08  0.22  0.13  0.07 -0.14
  -0.23  0.01  0.76]
 [-0.44  0.19 -0.16 -0.01  0.03  0.25 -0.05 -0.06  0.11  0.13  0.21 -0.18
  -0.46  0.02 -0.6 ]
 [-0.29  0.06 -0.03  0.18 -0.28  0.27  0.05 -0.24 -0.65  0.16 -0.46  0.
   0.08  0.03  0.06]
 [-0.13  0.14  0.08 -0.67  0.03  0.13  0.3  -0.25 -0.04 -0.57 -0.09  0.05
  -0.01  0.02  0.01]
 [-0.14 -0.01  0.68  0.02  0.07  0.06 -0.01  0.01  0.04  0.04  0.03 -0.65
   0.29 -0.01 -0.05]]
```

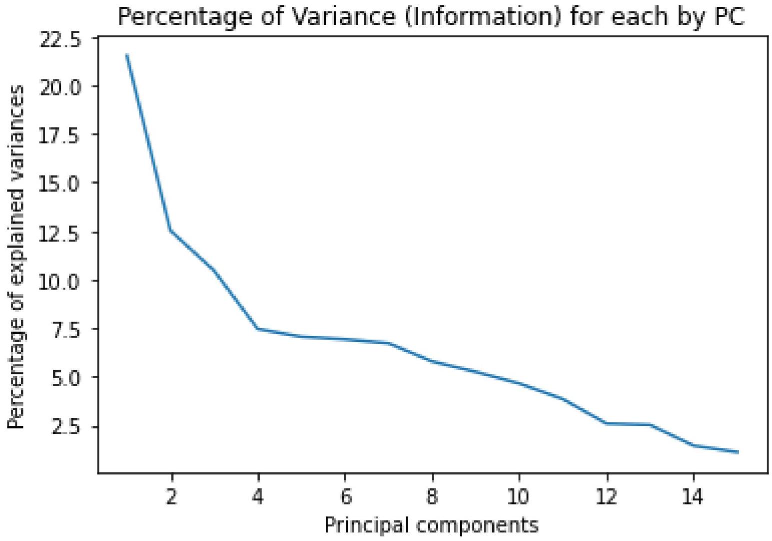
In [140]:

```
explained_variances2 = [i/np.sum(eigenValues2) for i in eigenValues2]

print("Sum of  Explained Variance :",np.sum(explained_variances2))
print("Explained Variance :", explained_variances2)
```

```
Sum of  Explained Variance : 0.9999999999999999
Explained Variance : [0.2151898734177215, 0.1252498334443704, 0.104596935376
41572, 0.07461692205196535, 0.07061958694203864, 0.0692871419053964, 0.06728
847435043304, 0.05796135909393737, 0.05263157894736842, 0.04663557628247834,
0.03864090606262491, 0.02598267821452364 7, 0.02531645569620252 8, 0.014656895
403064623, 0.011325782811459026]
```

In [143]:

```
#Plot the principal components and percentage of explained variances.
import matplotlib.pyplot as plt
X_axis2 = [i for i in range(1, 16)]
q2 = [i*100 for i in explained_variances2]

plt.plot(X_axis2,q2)
plt.xlabel('Principal components')
plt.ylabel('Percentage of explained variances')
plt.title('Percentage of Variance (Information) for each by PC')
plt.show()
```



In [146]:

```
#consider only the features upto where the pov adds to <95%
k2, sum2 = 0, 0
while sum2 < 95:
    sum2 += q2[k2]
    k2 += 1
k2 -= 1
print("k : ",k2)
eisub2 = eigenVectors2[:,:k2]
eisub2
```

```
k :  12
```

Out[146]:

```
array([[ 0.06,  0.37,  0.04,  0.51, -0.22,  0.12, -0.23, -0.19,  0.01,
        -0.59,  0.28,  0.01],
       [-0.3 , -0.09,  0.03,  0.12, -0.22, -0.49, -0.17,  0.06,  0.34,
        -0.26, -0.59, -0.02],
       [ 0.11, -0.01, -0.03, -0.03,  0.64,  0.22, -0.58, -0.3 ,  0.04,
        -0.05, -0.31,  0.01],
       [ 0.2 ,  0.59,  0.05, -0.1 ,  0.07, -0.16,  0.06,  0.1 ,  0.03,
         0.23, -0.19, -0.01],
       [ 0.17,  0.63,  0.04, -0.03,  0.  , -0.13,  0.02,  0.03, -0.01,
         0.12, -0.13, -0.01],
       [-0.21,  0.04, -0.05,  0.16,  0.49, -0.18,  0.13,  0.57, -0.45,
        -0.32,  0.02, -0.03],
       [-0.07, -0.02, -0.02,  0.38,  0.38, -0.23,  0.59, -0.53,  0.13,
         0.06, -0.01,  0.01],
       [-0.43,  0.16, -0.12,  0.  ,  0.09,  0.1 , -0.04,  0.09,  0.22,
         0.08,  0.08,  0.35],
       [-0.13, -0.01,  0.68,  0.06,  0.04,  0.05, -0.03,  0.03, -0.01,
         0.1 ,  0.05,  0.64],
       [-0.19,  0.02, -0.  , -0.23, -0.02, -0.62, -0.34, -0.35, -0.36,
         0.1 ,  0.38,  0.01],
       [-0.48,  0.15, -0.1 , -0.05,  0.04,  0.07, -0.04,  0.08,  0.22,
         0.13,  0.07, -0.14],
       [-0.44,  0.19, -0.16, -0.01,  0.03,  0.25, -0.05, -0.06,  0.11,
         0.13,  0.21, -0.18],
       [-0.29,  0.06, -0.03,  0.18, -0.28,  0.27,  0.05, -0.24, -0.65,
         0.16, -0.46,  0.  ],
       [-0.13,  0.14,  0.08, -0.67,  0.03,  0.13,  0.3 , -0.25, -0.04,
        -0.57, -0.09,  0.05],
       [-0.14, -0.01,  0.68,  0.02,  0.07,  0.06, -0.01,  0.01,  0.04,
         0.04,  0.03, -0.65]])
```

In [147]:

```
#splitting the dataset into training and testing sets using sklearn.model_selection
from sklearn.model_selection import train_test_split as split_data

train_X, test_X, train_Y, test_y = split_data(X, Y, train_size = .75, shuffle=True)
train_X.shape, test_X.shape, train_Y.shape, test_y.shape
```

Out[147]:

```
((2742, 15), (914, 15), (2742,), (914,))
```

In [ ]:

```
# Logistic Regression
# Implement the logic of the algorithm using Gradient Descent Function
# Estimate linear regression coefficients using stochastic gradient descent

from math import exp

# Make a prediction with coefficients

def sigmoid(z):
    return 1.0 / (1.0 + exp(-z))

def predict(row, coeff):
    y_pred = coeff[0]
    for i in range(len(row)):
        y_pred += coeff[i + 1] * row[i]
    return sigmoid(y_pred)

def Gradient_Descent(x, y, alpha, epochs):
    coef = [0.0]*(len(x)+1)
    for epoch in range(epochs):
        for i in range(0, len(x)-1):
            y_pred = predict(x[i], coef)
            error = y[i] - y_pred
            coef[0] = coef[0] + alpha*error*y_pred*(1.0 - y_pred)
            for j in range(len(x[i])):
                coef[j + 1] = coef[j + 1] + alpha*error*y_pred*(1.0 - y_pred)*x[i][j]
    return coef

alpha = 0.1
epochs = 500

# Finding coefficients
coef = Gradient_Descent(train_X, train_Y, alpha, epochs)
print(np.around(coef,4))
```

In [ ]:

```
# Predict the values using test data
Y_pred = []
for i in range(len(test_X)):
    y = predict(test_X[i],coef)
    Y_pred.append(y)

# print predicted value
print("Predicted Value for testing data")
print(np.around(Y_pred,3))


# To calculate LOSS
def LOG_LOSS(actual, predict):
    error = 0.0
    for i in range(len(actual)):
        pred_error_0 = actual[i] * np.log(predict[i])
        pred_error_1 = (1 - actual[i]) * np.log(1 - predict[i])
        error += pred_error_0 + pred_error_1
    mean_error = -error/float(len(actual))
    return mean_error

mean_loss = LOG_LOSS(Y_test, Y_pred)
print("\nMean Error :- ", mean_loss)
```