In [1]:
```
#PCA for diabetes dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#load dataset
df = pd.read_csv("diabetes.csv")
```

In [3]:
```
column_names = df.columns.tolist()
print (column_names)
```

```
['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
```

In [4]:
```
#setup (X,Y): X: Features, Y: Target or Outcome
X = df.drop(['Outcome'], axis=1)
Y = df['Outcome']
```

In [6]:
```
X.columns.tolist()
```

Out[6]:
```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age']
```

In [8]:
```
#Standardize and Scale the Feature Data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
scaled_data = scaler.transform(X)
```

Principal Component Analysis

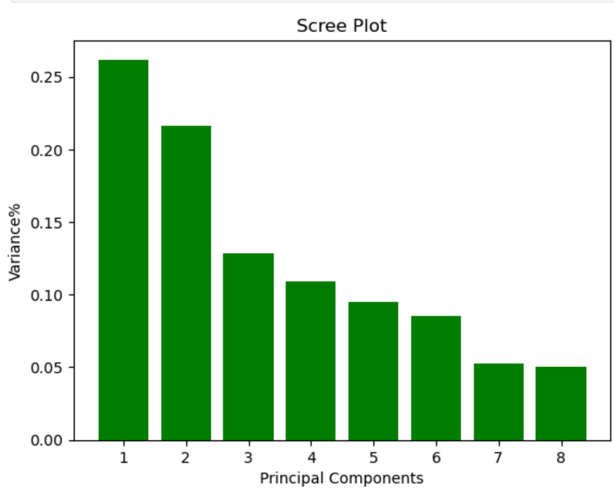
In [19]:
```
from sklearn.decomposition import PCA
#define PCA model to use. Pick number of components to
pca = PCA(n_components=None) # include all components
#pca = PCA(n_components=6) # top 4 components

#fit PCA model to data
pca.fit(scaled_data)
```

Out[19]:
```
▾ PCA
PCA()
```

In [20]:
```
#Create SCREE plot
#provides total variance contributed by each PC

PCA_values = range(1, pca.n_components_+1)
plt.bar(PCA_values, pca.explained_variance_ratio_, color='green')
#PCA_values = np.arange(pca.n_components_) + 1
#plt.plot(PCA_values, pca.explained_variance_ratio_, 'o-', linewidth=2, color='blue')
plt.title('Scree Plot')
plt.xlabel('Principal Components')
plt.ylabel('Variance%')
plt.xticks(PCA_values)
plt.show()
```



In [22]:
```
print (pca.explained_variance_ratio_)
```

```
[0.26179749 0.21640127 0.12870373 0.10944113 0.09529305 0.08532855
 0.05247702 0.05055776]
```

Find the loadings for the PCs

In [23]:
```
PCnames = ['PC'+str(i+1) for i in range(pca.n_components_)]
```

```
print (PCnames)
```

['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8']

In [24]:
```
Loadings = pd.DataFrame(pca.components_, columns=PCnames, index=X.columns)
```

In [25]:
```
Loadings.iloc[:,:2]
```

Out[25]:

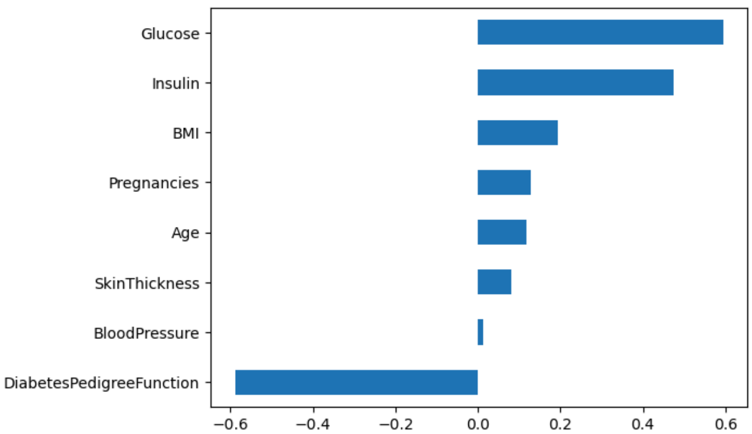
	PC1	PC2
Pregnancies	0.128432	0.393083
Glucose	0.593786	0.174029
BloodPressure	0.013087	-0.467923
SkinThickness	0.080691	-0.404329
Insulin	0.475606	-0.466328
BMI	0.193598	0.094162
DiabetesPedigreeFunction	-0.588790	-0.060153
Age	0.117841	0.450355

In [ ]:
```
#Note. High glucose will have higher scores in PC1 and
#high DPF will have low scores on PC1.
```

Influencers of PC1

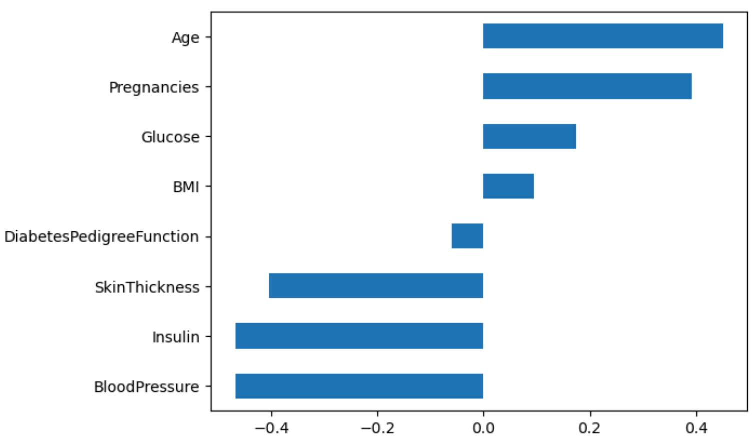
In [26]:
```
Loadings["PC1"].sort_values().plot.barh()
#barh. for Horizontal bar plot.
```

Out[26]: <Axes: >



In [66]:
```
Loadings["PC2"].sort_values().plot.barh()
```

Out[66]: <Axes: >



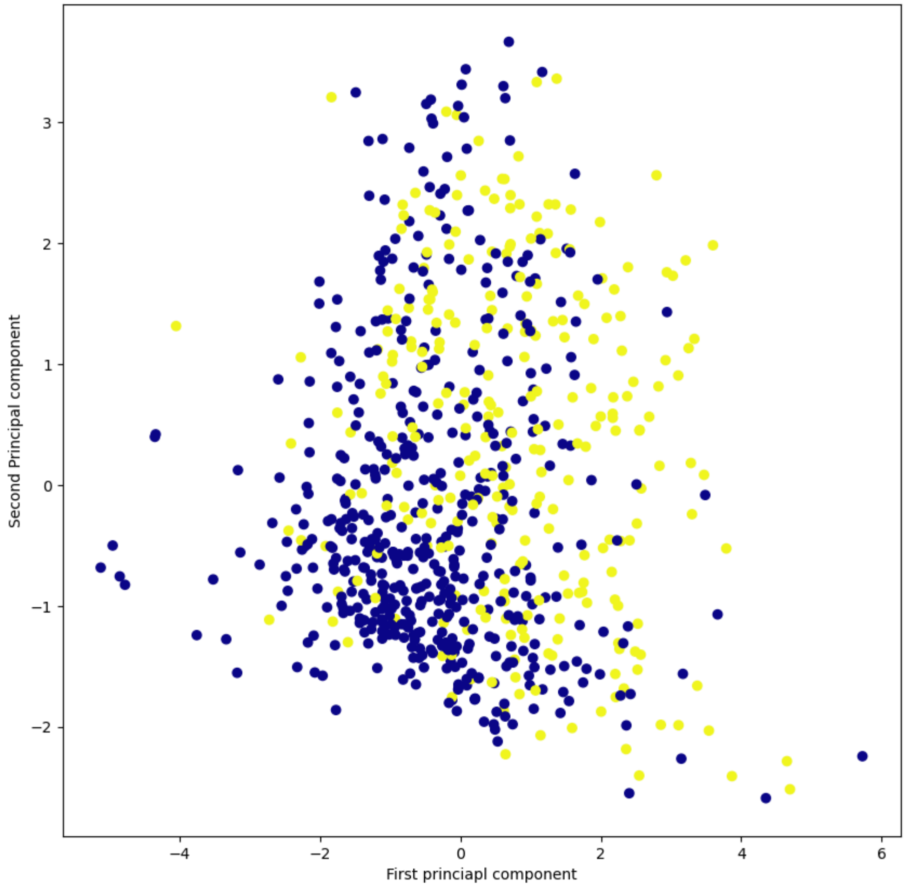
Under the Hood Computations

In [38]:
```
#Transform the data into principal components
X_pca = pca.transform(scaled_data)
```

In [39]:
```
#How many principal components we got?
pca.components_
```

In [25]:
```
#Display data
plt.figure(figsize=(10,10))
plt.scatter(X_pca[:,0], X_pca[:,1], c=df['Outcome'], cmap='plasma')
plt.xlabel('First princiapl component')
plt.ylabel('Second Principal component')
```

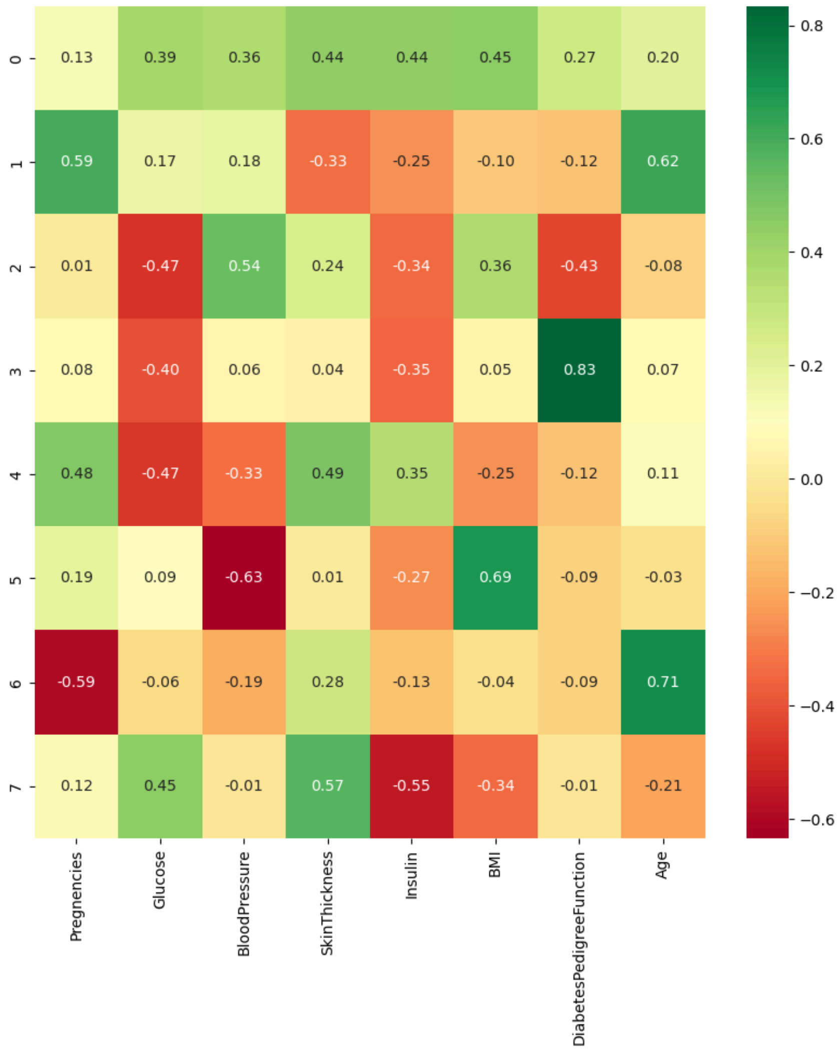
Out[25]: Text(0, 0.5, 'Second Principal component')



Heatmap with Raw Data

In [13]:
```
df_comp = pd.DataFrame(pca.components_, columns=["Pregnencies", "Glucose",
                                                 "BloodPressure","SkinThickness",
                                                      "Insulin","BMI",
                                                      "DiabetesPedigreeFunction",
                                                      "Age"])
plt.figure(figsize=(10,10))
#sns.heatmap(df_comp, annot=True, cmap='plasma')
sns.heatmap(df_comp, annot=True, fmt='.2f', cmap='RdYlGn')
```

Out[13]: <Axes: >



Standardize data for PCA

In [15]:
```
#PCA requires Data Standardization.
#Shift the distribution to zero mean and a std deviation equal to 1

from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X)
print (X_std)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

In [74]:
```
#Calculate Covarience Matrix
#mean_vec = np.mean(X_std, axis=0)
#cov_mat = (X_std - mean_vec).T.dot(X_std - mean_vec)/(X_std.shape[0]-1)
#print ("Covarience Matrix \n %s" %cov_mat)
```

Covariance Matrix using numpy

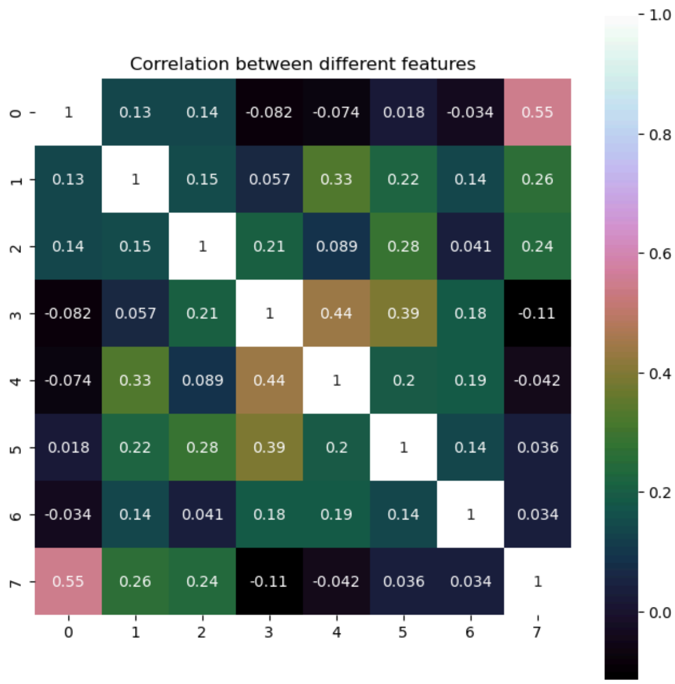
In [17]:
```
#Now calculate Numpy covarience Matrix
cov_mat = np.cov(X_std.T)
print ("Numpy Covarience Matrix \n %s" %(np.cov(X_std.T)))
```

```
Numpy Covarience Matrix
 [[ 1.00130378  0.12962746  0.14146618 -0.08177826 -0.07363049  0.01770615
  -0.03356638  0.54505093]
 [ 0.12962746  1.00130378  0.15278853  0.05740263  0.33178913  0.2213593
   0.13751636  0.26385788]
 [ 0.14146618  0.15278853  1.00130378  0.2076409   0.08904933  0.2821727
   0.04131875  0.23984024]
 [-0.08177826  0.05740263  0.2076409   1.00130378  0.43735204  0.39308503
   0.18416737 -0.11411885]
 [-0.07363049  0.33178913  0.08904933  0.43735204  1.00130378  0.19811702
   0.18531222 -0.04221793]
 [ 0.01770615  0.2213593   0.2821727   0.39308503  0.19811702  1.00130378
   0.14083033  0.03628912]
 [-0.03356638  0.13751636  0.04131875  0.18416737  0.18531222  0.14083033
   1.00130378  0.03360507]
 [ 0.54505093  0.26385788  0.23984024 -0.11411885 -0.04221793  0.03628912
   0.03360507  1.00130378]]
```

Generate heatmap for Coverance Matrix

In [69]:
```python
plt.figure(figsize=(8,8))
sns.heatmap(cov_mat, vmax=1, square=True, annot=True, cmap='cubehelix')
plt.title("Correlation between different features")
```

Out[69]: `Text(0.5, 1.0, 'Correlation between different features')`



In [19]:
```python
#Find eigenvalues and eigenvectors from the Covarience Matrix
eigen_values, eigen_vectors = np.linalg.eig(cov_mat)
print ("Eigen vectors: Diection of main axes of the data (principical components) \n%s" %eigen_vectors)
print ("Eigen values \n%s" %eigen_values)
#greater the eigenvalue, greater is the variance
```

```
Eigen vectors: Diection of main axes of the data (principical components)
[[-0.1284321  -0.59378583 -0.58879003  0.11784098 -0.19359817  0.47560573
  -0.08069115  0.01308692]
 [-0.39308257 -0.17402908 -0.06015291  0.45035526 -0.09416176 -0.46632804
   0.40432871 -0.46792282]
 [-0.36000261 -0.18389207 -0.19211793 -0.01129554  0.6341159  -0.32795306
  -0.05598649  0.53549442]
 [-0.43982428  0.33196534  0.28221253  0.5662838  -0.00958944  0.48786206
  -0.03797608  0.2376738 ]
 [-0.43502617  0.25078106 -0.13200992 -0.54862138  0.27065061  0.34693481
   0.34994376 -0.33670893]
 [-0.45194134  0.1009598  -0.03536644 -0.34151764 -0.68537218 -0.25320376
  -0.05364595  0.36186463]
 [-0.27061144  0.122069   -0.08609107 -0.00825873  0.08578409 -0.11981049
  -0.8336801  -0.43318905]
 [-0.19802707 -0.62058853  0.71208542 -0.21166198  0.03335717  0.10928996
  -0.0712006  -0.07524755]]
Eigen values
[2.09711056 1.73346726 0.42036353 0.40498938 0.68351839 0.76333832
 0.87667054 1.03097228]
```

In [20]:
```python
#First, make a list of eigenvalues and eigenvector tuples
eigen_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:,i]) for i in range(len(eigen_values))]

#sort eigenvalue, eigenvector tuples from high to low
eigen_pairs.sort(key=lambda x: x[0], reverse=True)

#List them in revse order to visually check
print ("Eigenvalues in decending order: ")
for i in eigen_pairs:
    print(i[0])
```

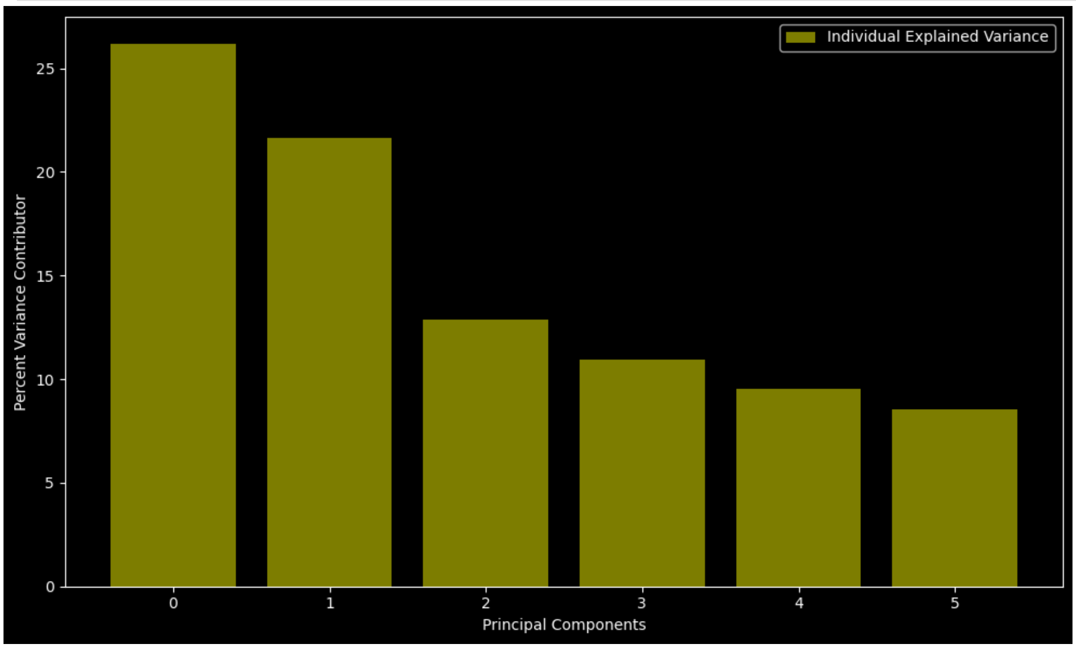
```
Eigenvalues in decending order:
2.097110557994524
1.7334672594471234
1.0309722810083821
0.8766705419094792
0.7633383156496728
0.6835183858447288
0.42036352804956845
0.4049893778148988
```

In [21]:
```
#Find big contributor (eigenvalues) to the variance
total = sum(eigen_values)
percent_variance = [(i/total)*100 for i in sorted(eigen_values, reverse=True)]
print (percent_variance)
```

```
[26.17974931611004, 21.640126757746494, 12.870373364801912, 10.944113047600439, 9.529304819389639, 8.532854849331173, 5.247702246321927, 5.055775598698365]
```

In [41]:
```
#note: 90% of variance is explained by seven eigenvalues. Last two can be dropped
```

In [22]:
```
#plot the variance
with plt.style.context('dark_background'):
    plt.figure(figsize=(10,6))
    plt.bar(range(6), percent_variance[:6], alpha=0.5, align='center', label='Individual Explained Variance', color='yellow')
    plt.ylabel('Percent Variance Contributor')
    plt.xlabel('Principal Components')
    plt.legend(loc='best')
    plt.tight_layout()
```



In [23]:
```
#Projection Matrix with Reduced set of feautures
matrix_W = np.hstack((eigen_pairs[0][1].reshape(8,1), eigen_pairs[1][1].reshape(8,1), eigen_pairs[2][1].reshape(8,1),
                      eigen_pairs[3][1].reshape(8,1), eigen_pairs[4][1].reshape(8,1),
                       eigen_pairs[5][1].reshape(8,1)))

print ("Matrix W:\n", matrix_W)
```

```
Matrix W:
 [[-0.59378583 -0.59378583 -0.59378583 -0.59378583 -0.59378583 -0.59378583]
 [-0.17402908 -0.17402908 -0.17402908 -0.17402908 -0.17402908 -0.17402908]
 [-0.18389207 -0.18389207 -0.18389207 -0.18389207 -0.18389207 -0.18389207]
 [ 0.33196534  0.33196534  0.33196534  0.33196534  0.33196534  0.33196534]
 [ 0.25078106  0.25078106  0.25078106  0.25078106  0.25078106  0.25078106]
 [ 0.1009598   0.1009598   0.1009598   0.1009598   0.1009598   0.1009598 ]
 [ 0.122069    0.122069    0.122069    0.122069    0.122069    0.122069  ]
 [-0.62058853 -0.62058853 -0.62058853 -0.62058853 -0.62058853 -0.62058853]]
```

In [24]:
```
#Project onto new feature space. Recuded to five features from eight
# Y = X (times) W
Y = X_std.dot(matrix_W)
Y
```

Out[24]:
```
array([[-1.23489499, -1.23489499, -1.23489499, -1.23489499, -1.23489499,
        -1.23489499],
       [ 0.73385167,  0.73385167,  0.73385167,  0.73385167,  0.73385167,
         0.73385167],
       [-1.59587594, -1.59587594, -1.59587594, -1.59587594, -1.59587594,
        -1.59587594],
       ...,
       [-0.09706503, -0.09706503, -0.09706503, -0.09706503, -0.09706503,
        -0.09706503],
       [-0.83706234, -0.83706234, -0.83706234, -0.83706234, -0.83706234,
        -0.83706234],
       [ 1.15175485,  1.15175485,  1.15175485,  1.15175485,  1.15175485,
         1.15175485]])
```

In [ ]:
```
END
```