In [1]:
```
#Problem Statement

#In this assignment students have to transform iris data into 3 dimensions and plo
#and color each data point with specific class.
```

In [2]:
```
# Import libraries into working environment:
```

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import decomposition
from sklearn import datasets
import seaborn as sns
from sklearn.decomposition import PCA
```

In [4]:
```
# Load iris data set:
```

In [5]:
```
iris = datasets.load_iris()
X = iris.data
y = iris.target
print("Number of samples:")

print(X.shape[0])
print('-------------------------------------------------------------------------------
print('Number of features :')
print(X.shape[1])
print('-------------------------------------------------------------------------------
print("Feature names:")
print('-------------------------------------------------------------------------------
print(iris.feature_names)
```

```
Number of samples:
150
-------------------------------------------------------------------------------
------
Number of features :
4
-------------------------------------------------------------------------------
------
Feature names:
-------------------------------------------------------------------------------
------
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (c
m)']
```

In [6]:
```
# Feature scaling prior to applying PCA:
```

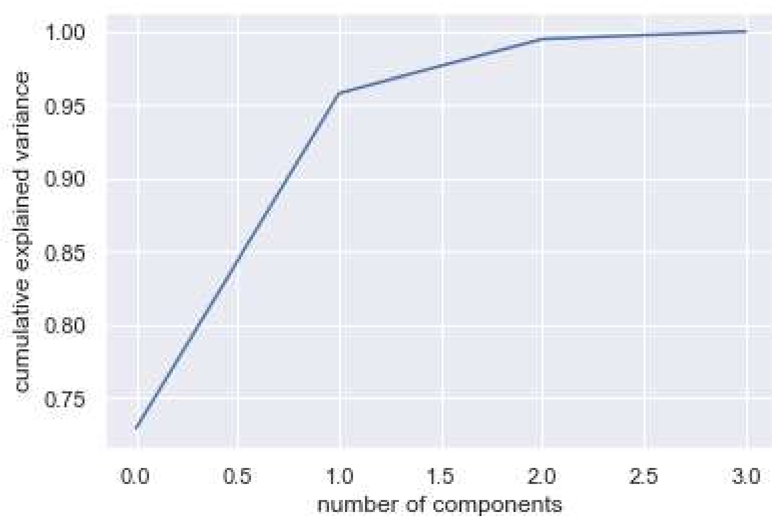
In [7]:
```
# Feature Scaling:

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_scaled = sc.fit_transform(X)
print('Shape of scaled data points:')
print('----------------------------------------------------------------------')
print(X_scaled.shape)
print('----------------------------------------------------------------------')
print('First 5 rows of scaled data points :')
print('----------------------------------------------------------------------')
print(X_scaled[:5, :])
```

```
Shape of scaled data points:
----------------------------------------------------------------------
(150, 4)
----------------------------------------------------------------------
First 5 rows of scaled data points :
----------------------------------------------------------------------
[[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
 [-1.14301691 -0.13197948 -1.34022653 -1.3154443 ]
 [-1.38535265  0.32841405 -1.39706395 -1.3154443 ]
 [-1.50652052  0.09821729 -1.2833891  -1.3154443 ]
 [-1.02184904  1.24920112 -1.34022653 -1.3154443 ]]
```

In [8]:
```
# Looking at the explained variance as a function of the components:
```

In [9]:
```
sns.set()
pca = PCA().fit(X_scaled)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



In [10]:
```
# Here we see that we'd need about 3 components to retain 100% of the variance.
#Looking at this plot for a high-dimensional dataset can help us understand the le
#observations.
```

In [11]:
```
# PCA using Eigen-decomposition: 5-step process:
```

In [12]:
```
# 1. Normalize columns of A so that each feature has zero mean:

A0 = iris.data
mu = np.mean(A0,axis=0)
A = A0 - mu
print("Does A have zero mean across rows?")
print(np.mean(A,axis=0))
print('---------------------------------------------------------------------------'
print('Mean value : ')
print('---------------------------------------------------------------------------'
print(mu)
print('Standardized Feature value first 5 rows: ')
print('---------------------------------------------------------------------------'
print(A[:5,:])

# 2. Compute sample covariance matrix Sigma = {A^TA}/{(m-1)}
#covariance matrix can also be computed using np.cov(A.T):

m,n = A.shape
Sigma = (A.T @ A)/(m-1)
print("---------------------------------------------------------------------------"
print("Sigma:")
print(Sigma)

# 3. Perform eigen-decomposition of Sigma using `np.linalg.eig(Sigma):

W,V = np.linalg.eig(Sigma)
print("---------------------------------------------------------------------------
print("Eigen values:")
print(W)
print("---------------------------------------------------------------------------
print("Eigen vectors:")
print(V)

# 4. Compress by ordering 3 eigen vectors according to largest eigen values and co

print("---------------------------------------------------------------------------
print("Compressed - 4D to 3D:")
print("---------------------------------------------------------------------------
print('First 3 eigen vectors :')
print(V[:,:3] )
print("---------------------------------------------------------------------------
Acomp = A @ V[:,:3]
print('First first five rows of transformed features :')
print("---------------------------------------------------------------------------
print(Acomp[:5,:])


# 5. Reconstruct from compressed version by computing $A V_k V_k^T$:

print("---------------------------------------------------------------------------
print("Reconstructed version - 3D to 4D:")
print("---------------------------------------------------------------------------
Arec = A @ V[:,:3] @ V[:,:3].T # first 3 evectors
print(Arec[:5,:]+mu) # first 5 obs, adding mu to compare to original
```

```
Does A have zero mean across rows?
[-1.12502600e-15 -7.60872846e-16 -2.55203266e-15 -4.48530102e-16]
--------------------------------------------------------------------------
Mean value :
--------------------------------------------------------------------------
[5.84333333 3.05733333 3.758      1.19933333]
Standardized Feature value first 5 rows:
--------------------------------------------------------------------------
[[-0.74333333  0.44266667 -2.358      -0.99933333]
 [-0.94333333 -0.05733333 -2.358      -0.99933333]
 [-1.14333333  0.14266667 -2.458      -0.99933333]
 [-1.24333333  0.04266667 -2.258      -0.99933333]
 [-0.84333333  0.54266667 -2.358      -0.99933333]]
--------------------------------------------------------------------------
Sigma:
[[ 0.68569351 -0.042434    1.27431544  0.51627069]
 [-0.042434    0.18997942 -0.32965638 -0.12163937]
 [ 1.27431544 -0.32965638  3.11627785  1.2956094 ]
 [ 0.51627069 -0.12163937  1.2956094   0.58100626]]
---------------------------------------------------------------------------
Eigen values:
[4.22824171 0.24267075 0.0782095  0.02383509]
---------------------------------------------------------------------------
Eigen vectors:
[[ 0.36138659 -0.65658877 -0.58202985  0.31548719]
 [-0.08452251 -0.73016143  0.59791083 -0.3197231 ]
 [ 0.85667061  0.17337266  0.07623608 -0.47983899]
 [ 0.3582892   0.07548102  0.54583143  0.75365743]]
----------------------------------------------------------------------------
Compressed - 4D to 3D:
----------------------------------------------------------------------------
First 3 eigen vectors :
[[ 0.36138659 -0.65658877 -0.58202985]
 [-0.08452251 -0.73016143  0.59791083]
 [ 0.85667061  0.17337266  0.07623608]
 [ 0.3582892   0.07548102  0.54583143]]
----------------------------------------------------------------------------
First first five rows of transformed features :
----------------------------------------------------------------------------
[[-2.68412563 -0.31939725 -0.02791483]
 [-2.71414169  0.17700123 -0.21046427]
 [-2.88899057  0.14494943  0.01790026]
 [-2.74534286  0.31829898  0.03155937]
 [-2.72871654 -0.32675451  0.09007924]]
----------------------------------------------------------------------------
Reconstructed version - 3D to 4D:
----------------------------------------------------------------------------
[[5.09928623 3.50072335 1.40108561 0.1982949 ]
 [4.86875839 3.03166108 1.4475168  0.12536791]
 [4.69370023 3.20638436 1.30958161 0.18495067]
 [4.6238432  3.07583667 1.46373578 0.25695828]
 [5.0193263  3.58041421 1.37060574 0.24616799]]
```

In [13]:

```python
# Original iris feature values:
```

In [14]:
```
iris.data[:5, :]
```

Out[14]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

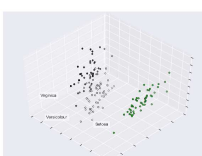
In [15]:
```
# 3D Visualization:
```

In [16]:
```
np.random.seed(5)

centers = [[1, 1], [-1, -1], [1, -1]]
fig = plt.figure(1, figsize=(8, 6))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, 1, 1], elev=48, azim=134)
y= iris.target
plt.cla()
for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:
    ax.text3D(Acomp[y == label, 0].mean(),
              Acomp[y == label, 1].mean() + 1.5,
              Acomp[y == label, 2].mean(), name,
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(Acomp[:, 0], Acomp[:, 1], Acomp[:, 2], c=y, cmap=plt.cm.nipy_spectral,
           edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])

plt.show()
```



In [17]:
```
# Applying PCA for number of compents = 3 using sklearn:
```

In [18]:
```
pca = PCA(n_components=3)
pca.fit(X_scaled)
print('explained variance :')
print('---------------------------------------------------------------------')
print(pca.explained_variance_)
print('---------------------------------------------------------------------')
print('PCA Components : ')
print('---------------------------------------------------------------------')
print(pca.components_)
print('---------------------------------------------------------------------')
X_transformed = pca.transform(X)
print('Transformed Feature values first five rows :')
print('---------------------------------------------------------------------')
print(X_transformed[:5,:])
print('---------------------------------------------------------------------')
print('Transformed Feature shape :')
print('---------------------------------------------------------------------')
print(X_transformed.shape)
print('---------------------------------------------------------------------')
print('Original Feature shape :')
print('---------------------------------------------------------------------')
print(X.shape)
print('---------------------------------------------------------------------')
print('Retransformed  Feature  :')
print('---------------------------------------------------------------------')

X_retransformed = pca.inverse_transform(X_transformed)

print('Retransformed Feature values first five rows :')
print('---------------------------------------------------------------------')
print(X_retransformed[:5,:])
```

```
explained variance :
---------------------------------------------------------------------
[2.93808505 0.9201649  0.14774182]
---------------------------------------------------------------------
PCA Components :
---------------------------------------------------------------------
[[ 0.52106591 -0.26934744  0.5804131   0.56485654]
 [ 0.37741762  0.92329566  0.02449161  0.06694199]
 [-0.71956635  0.24438178  0.14212637  0.63427274]]
---------------------------------------------------------------------
Transformed Feature values first five rows :
---------------------------------------------------------------------
[[ 2.64026976  5.2040413  -2.48862071]
 [ 2.6707303   4.66690995 -2.46689833]
 [ 2.45460631  4.77363639 -2.28832134]
 [ 2.54551709  4.64846339 -2.2123776 ]
 [ 2.56122842  5.2586291  -2.39222589]]
---------------------------------------------------------------------
Transformed Feature shape :
---------------------------------------------------------------------
(150, 3)
---------------------------------------------------------------------
Original Feature shape :
---------------------------------------------------------------------
```

```
(150, 4)
-------------------------------------------------------------------
Retransformed  Feature  :
-------------------------------------------------------------------
Retransformed Feature values first five rows :
-------------------------------------------------------------------
[[5.13057916 3.48554528 1.30620386 0.26127823]
 [4.92809758 2.98671832 1.31381567 0.25630533]
 [4.72726519 3.18711179 1.21636888 0.25463729]
 [4.67274664 3.06561278 1.2768626  0.34577854]
 [5.04063335 3.58079268 1.27536442 0.28142603]]
```

In [19]:
```
# Note:

#Transformed from 4D to 3D using PCA
```

In [20]:
```
print('First Principal Component PC1: ', pca.components_[0])
print('\nSecond Principal Component PC2: ', pca.components_[1])
print('\nThird Principal Component PC3 :', pca.components_[2])
```

```
First Principal Component PC1:  [ 0.52106591 -0.26934744  0.5804131   0.564856
54]

Second Principal Component PC2:  [0.37741762 0.92329566 0.02449161 0.06694199]

Third Principal Component PC3 : [-0.71956635  0.24438178  0.14212637  0.634272
74]
```

In [21]:
```
# Note:

#Transforming from 3D to 4D
```

In [22]:
```
# 3D Visualization:
```

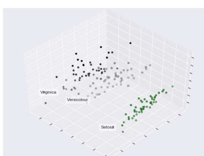
In [23]:
```
np.random.seed(5)
centers = [[1, 1], [-1, -1], [1, -1]]
fig = plt.figure(1, figsize=(8, 6))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, 1, 1], elev=48, azim=134)
y= iris.target
plt.cla()
for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:
    ax.text3D(X_transformed[y == label, 0].mean(),
              X_transformed[y == label, 1].mean() + 1.5,
              X_transformed[y == label, 2].mean(), name,
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

# Reorder the labels to have colors matching the cluster results

y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X_transformed[:, 0], X_transformed[:, 1], X_transformed[:, 2], c=y, cma
           edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])

plt.show()
```



In [ ]:
```
```