In [ ]:

```
#***************************************************************************
#
#                Assignment : To Do Dimension Reduction on Iris DataSet
#
#***************************************************************************
#Problem Statement: In this assignment students have to transform iris data into 3 dimensions
# and plot a 3d chart with transformed dimensions and color each data point with specific classxsa.

# Step 0 : importing packages
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


iris =  datasets.load_iris()

iris_df = pd.DataFrame(iris.data)

print(" Iris Data Set  4 dimentional - features\n",'-'*80)
print(iris_df.head())

print(" Iris Data labels - identification of type of flower\n",'-'*80)
print(iris.target)

print(" Iris Feature Columns \n",'-'*80)
print(iris.target_names)

print(" Description of the Features of the iris Dataset \n",'-'*80)
print(iris.DESCR)

#********************************************************************************************
```

In [ ]:

```
# Dimension Reduction by Eigen Decomposition

#Step1: Normaization of the features , to get mean as Zero

# mean of the Feaures of the Iris Df
mu =  np.mean(iris_df)

print(" The mean of each feature in the Iris Dataset\n",'-'*80)
print(mu)

iris_df = iris_df-mu

print(" The Iris DataSet after normalised by the mean \n",'-'*80)
print(iris_df.head())

# check if the mean of the features after normalisation if it is zero
print(iris_df.mean())

```

In [ ]:

```
# Step 2 :Computing the Covariance matrix $\Sigma = {A^TA}/{(m-1)}$

m,n = iris_df.shape
#print(m,n)

Sigma = (iris_df.T@iris_df)/(m-1)

print(" The Covariance Matrix - Sigma \n",'-'*80)
print(Sigma)
```

In [ ]:

```
#Step 3:  Perform eigen-decomposition of $\Sigma$ using `np.linalg.eig(Sigma)`
l,X = np.linalg.eig(Sigma)
print("---")

print( " The Eigen Values and Vectors, after Eigen Decomposition \n",'-'*80)
print("Evalues:")
print(l)
print("---")

print("Evectors:")
print(X)
```

In [ ]:

```
# Step 4. Compress by ordering $k$ evectors according to largest evalues and compute $AX_k$

print("---")
print("Iris DataSet Compressed - 4D to 3D:\n",'-'*80)
iris_comp = iris_df @ X[:,:3] # first 3 evectors (0,1,2)

print(iris_comp.head()) # first 5 observations

```

In [ ]:

```
# Plotting the 3D chart with compressed DataSet


# Fixing random state for reproducibility
np.random.seed(19680801)



fig = plt.figure(figsize=(9,6))
ax = Axes3D(fig,elev=-150, azim=110)

y = np.arange(150)


ax.scatter(iris_comp[0], iris_comp[1],iris_comp[2], c=y, cmap=plt.cm.Set1,edgecolor='k',s=40)


ax.set_xlabel('Eigen Vector1')
ax.set_ylabel('Eigen Vector2')
ax.set_zlabel('Eigen Vector3')

plt.title(" Iris Compressed 3 D plot")

plt.show()
```

In [ ]:

```
# Step5 : Reconstruction of the iris Dataset to original dimension from compressed

print("---")
print("Reconstructed version - 3D to 4D:")
iris_rec = iris_df @ X[:,:3] @ X[:,:3].T # first 2 evectors

print(round((iris_rec+mu).head(),1)) # first 5 obs, adding mu to compare to original

```

In [ ]:

```
# Dimension Reduction by using PCA library


# importing the PCA library from sklearn decomposition
from sklearn.decomposition import PCA

pca = PCA(n_components=3) # threee components
pca.fit(iris.data) # run PCA, putting in raw version for fun

print("Principal components:\n",'-'*80)
print(pca.components_)

print("---")
#print(iris_comp.head())
print("Compressed - 4D to 3D:\n",'-'*80)
iris_pcacom = pca.transform(iris.data)

print(iris_pcacom.shape)
print(iris_pcacom[:5,:]) # first 5 obs

print("---")
print("Reconstructed - 3D to 4D:\n",'-'*80)
print(pca.inverse_transform(pca.transform(iris.data))[:5,:]) # first 5 obs



# the Variance ratio
print(" The variance ratio is :\n",'-'*80)
variance = pca.explained_variance_ratio_ #calculate variance ratios
print(variance)

var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=3)*100)
print(var) #cumulative sum of variance explained with [n] features


fig = plt.figure( figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)

#y = iris.target

#X_reduced = PCA(n_components=3).fit_transform(iris.data)
```

```
ax.scatter(iris_pcacom[:,0], iris_pcacom[:,1], iris_pcacom[:,2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)

ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])
```