Experiment 4

AIM : Principal Component Analysis

To Perform PCA on given dataset and reduce its dimensionity.

DESCRIPTION:

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. In Machine Learning, this is used to reduce dimentionality of data. It is useful because it makes data simpler and more interpretable and visualization of data is also easy. In this, dimensions are reduced without much loss of data.

Steps in PCA :

1. Take whole dataset consisting of d-dimensional samples ignoring class label
2. Compute d-dimensional mean vector
3. Computer Covariance Matrix
4. Compute Eigen Vectors and corresponding Eigen Values
5. Sort Eigen Vectors with decreasing Eigen Values 6. Choose k Eigen Vectors with largest Eigen Values to form d*k dimensional matrix (where every column represents an eigen vector
6. Use this d*k matrix to transform d dimensional data to k dimensions #### CODE and OUTPUT :

In [1]:
```
# Principal Component Analysis
# Libraries
import numpy as np
import scipy as sp
from sklearn import datasets
import math
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# We are using IRIS dataset provided by sklearn
iris = datasets.load_iris()
X = iris.data
y = iris.target
print(X.shape,y.shape)
```

```
(150, 4) (150,)
```

In [3]:
```
# Calculation of Mean Vector
meanVector = np.mean(X,axis=0)
print(meanVector)
```

```
[5.84333333 3.054      3.75866667 1.19866667]
```

In [4]:
```
# Calculation of Covariance matrix
Xnew = X - meanVector
cov = (np.matmul(np.transpose(Xnew),Xnew)) / Xnew.shape[0]
print(cov)
```

```
[[ 0.68112222 -0.03900667  1.26519111  0.51345778]
 [-0.03900667  0.18675067 -0.319568   -0.11719467]
 [ 1.26519111 -0.319568    3.09242489  1.28774489]
 [ 0.51345778 -0.11719467  1.28774489  0.57853156]]
```

In [5]:
```
# Calculating Eigen Values and Eigen Vectors
eVal, eVec = np.linalg.eig(cov)
```

In [6]:
```
# Sorting Eigen Vectors according to decreasing Eigen Values
idx = eVal.argsort()[::-1]
eValSorted = eVal[idx]
eVecSorted = eVec[:, idx]
print(eValSorted)
print(eVecSorted)
```

```
[4.19667516 0.24062861 0.07800042 0.02352514]
[[ 0.36158968 -0.65653988 -0.58099728  0.31725455]
 [-0.08226889 -0.72971237  0.59641809 -0.32409435]
 [ 0.85657211  0.1757674   0.07252408 -0.47971899]
 [ 0.35884393  0.07470647  0.54906091  0.75112056]]
```

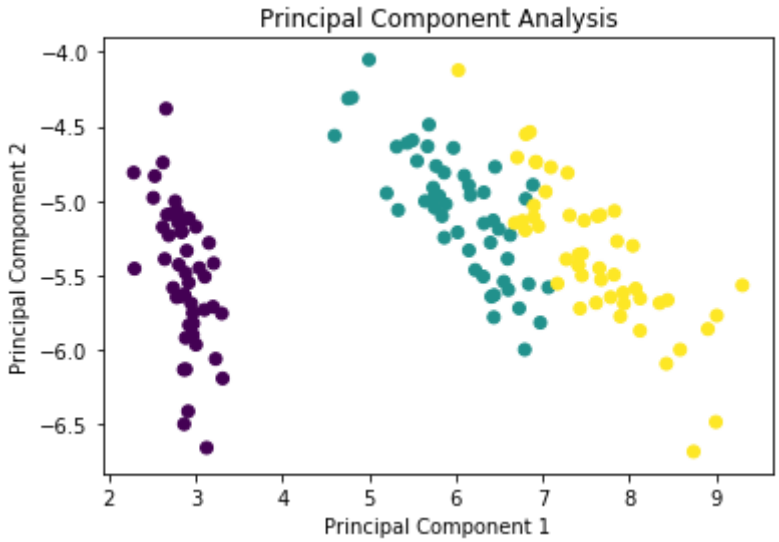
In [7]:
```
# Selecting Principal Components
pComps = eVecSorted[:, :2]
print(pComps)
```

```
[[ 0.36158968 -0.65653988]
 [-0.08226889 -0.72971237]
 [ 0.85657211  0.1757674 ]
 [ 0.35884393  0.07470647]]
```

In [8]:
```
# Transforming sample data to principal components
PCAdata = np.matmul(X,pComps)
print(PCAdata.shape)
```

```
(150, 2)
```

In [9]:
```
# Representation  of PCA Data
plt.scatter(PCAdata[:,0],PCAdata[:,1],c=y)
plt.title('Principal Component Analysis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Compoment 2')
plt.show()
```



Learning Outcomes :

In this experiment we learned how we can reduce dimensions of data and make it more readable and understandable by applying Principal Component Analysis