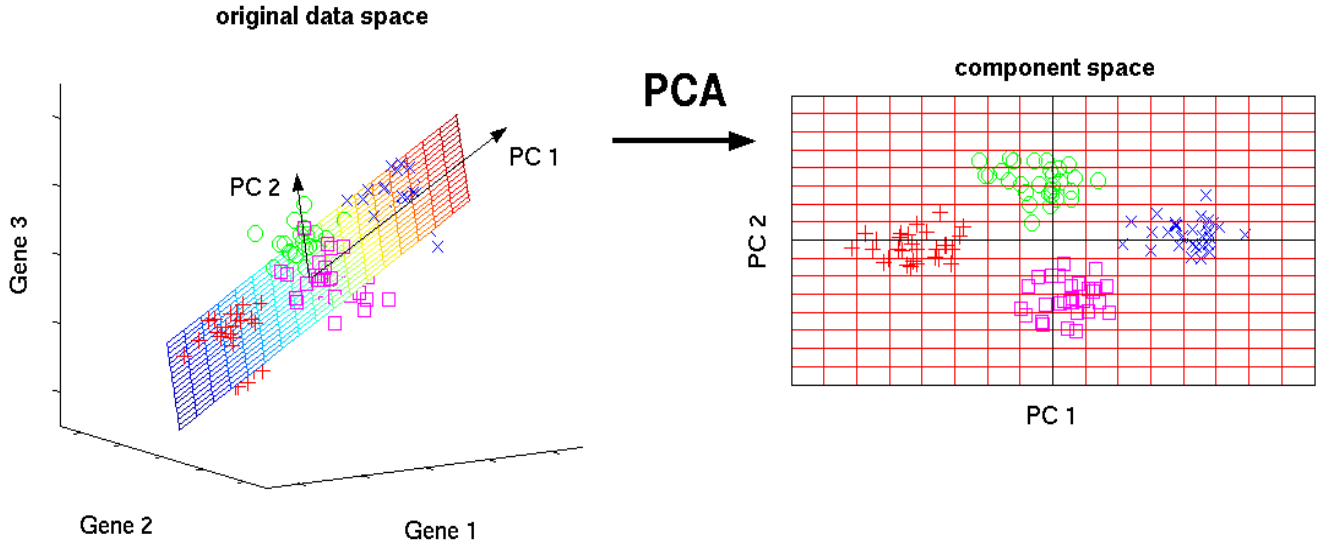
Experiment :

AIM : Write a Program to Implement Principle Component Analysis (PCA) algorithm.

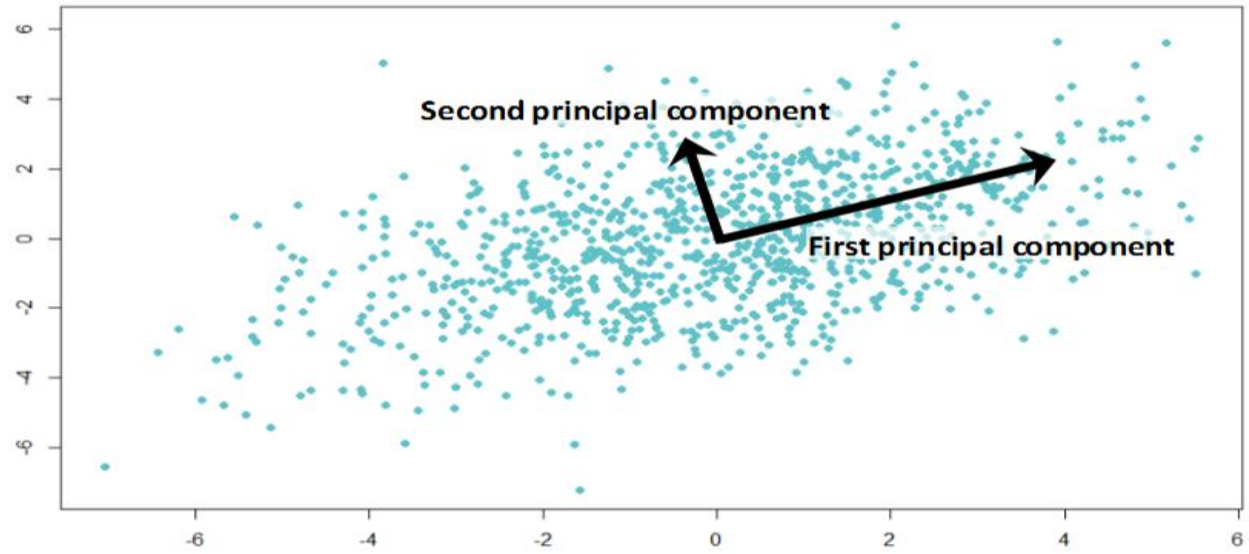
THEORY :

PCA is a method of factor analysis, which helps in finding a mapping from the inputs in the original d dimensional space to a new (k < d) dimensional space, with minimum loss of information.
It is a way of identifying patterns in data, and expressing the data in such a way as to highlight their similarities and differences.



Principal component analysis (PCA) simplifies the complexity in high-dimensional data while retaining trends and patterns. It does this by transforming the data into fewer dimensions, which act as summaries of features. High-dimensional data are very common in biology and arise when multiple features, such as expression of many genes, are measured for each sample. This type of data presents several challenges that PCA mitigates: computational expense and an increased error rate due to multiple test correction when testing each feature for association with an outcome. PCA is an unsupervised learning method and is similar to clustering—it finds patterns without reference to prior knowledge about whether the samples come from different treatment groups or have phenotypic differences.

PCA reduces data by geometrically projecting them onto lower dimensions called principal components (PCs), with the goal of finding the best summary of the data using a limited number of PCs.



Standard Deviation and Variance: The standard deviation (s) of a sample is the average distance from the mean of the dataset to a point.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(X_i - X')^2}{n-1}}$$

Like standard deviation, it is another measure for the spread of data.

$$s^2 = \frac{\sum_{i=1}^{n}(X_i - X')^2}{n-1}$$

Here, n is the number of elements in the sample, Xi is the i th element and X' is the mean.

Covariance: Standard deviation and variance are one dimensional measures. For each dimension of the data set, they can give results independently of the other dimensions. Covariance is always measuredbetween2 dimensions.
Covariance can be computed as follows:

$$Cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - X')(Y_i - Y')}{n-1}$$

Here, X' and Y' are means of X and Y respectively.
The covariance matrix for a 3-dimensional dataset with three dimensions x, y and z is as follows:

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

Eigen Vectors: Eigenvector is a vector which when operated on by a given operator gives a scalar multiple of itself. Eigenvectors can only be found for square matrices. But not every square matrix has eigenvectors.

Eigen Value: Eigenvalue associated with an eigenvector. It is the amount by which the original vector was scaled after multiplication by the square matrix.

ALGORITHM :

Step **1**: Get some data.

Step **2**: Subtract the mean.

Step **3**: Calculate the covariance matrix.

Step **4**: Calculate the eigenvectors and eigenvalues of the covariance matrix.

Step **5**: Sort the eigenvectors corresponding to eigenvalues in descending order and choose components(k<d) and forming a feature vector.

$$FeatureVector = (eig_1\ eig_2\ eig_3\ ....\ eig_n)$$

Step **6**: Deriving the new data set.

$$FinalData = RowFeatureVector \times RowDataAdjust,$$

CODE:

```
In [1]: import pandas as pd

        df = pd.read_csv('transfusion.csv')

In [2]: df.head()

In [3]: X = df.iloc[:,0:4].values
        y = df.iloc[:,4].values
        X.shape

Out[3]: (748, 4)

In [4]: y.shape

Out[4]: (748,)

In [5]: import plotly.plotly as py
        from plotly.graph_objs import *
        import plotly.tools as tls

        import plotly
        plotly.tools.set_credentials_file(username='srjssj', api_key='oRV6pGWR7OuxSwdQg49a')

In [6]: # plotting histograms
        traces = []
        legend = {0:False, 1:False, 2:False, 3:True}

colors = {0: 'rgb(31, 119, 180)',1: 'rgb(255, 127,14)',}
        for col in range(4):
            for key in colors:
                traces.append(Histogram(x=X[y==key, col],
                                        opacity=0.75,
                                        xaxis='x%s' %(col+1),
                                        marker=Marker(color=colors[key]),
                                        name=key,
                                        showlegend=legend[col]))

        data = Data(traces)
        layout = Layout(barmode='overlay',
                        xaxis=XAxis(domain=[0, 0.25], title='Recency'),
                        xaxis2=XAxis(domain=[0.3, 0.5], title='Frequency'),
                        xaxis3=XAxis(domain=[0.55, 0.75], title='Monetary'),
                        xaxis4=XAxis(domain=[0.8, 1], title='Time'),
                        yaxis=YAxis(title='Count'),
                        title='Distribution of Blood Donation')


        fig = Figure(data=data, layout=layout)
```

```
            py.iplot(fig)

In [7]: from sklearn.preprocessing import StandardScaler
            X_std = StandardScaler().fit_transform(X)

In [8]: import numpy as np
            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('Covariance matrix \n%s' %cov_mat)

In [9]: print('NumPy covariance matrix: \n%s' %np.cov(X_std.T))

In [10]: cov_mat = np.cov(X_std.T)

            eig_vals, eig_vecs = np.linalg.eig(cov_mat)

            print('Eigenvectors \n%s' %eig_vecs)
            print('\nEigenvalues \n%s' %eig_vals)

In [11]: # Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

            # Sort the (eigenvalue, eigenvector) tuples from high to low
            eig_pairs.sort()
            eig_pairs.reverse()

            # Visually confirm that the list is correctly sorted by decreasing eigenvalues
            print('Eigenvalues in descending order:')
for i in eig_pairs:
                print(i[0])


In [12]: tot = sum(eig_vals)
            var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)] cum_var_exp =
            np.cumsum(var_exp)

            trace1 = Bar(
                        x=['PC %s' %i for i in range(1,5)],
                        y=var_exp,
                        showlegend=False)
            trace2 = Scatter(
                        x=['PC %s' %i for i in range(1,5)],
                        y=cum_var_exp,
                        name='cumulative explained variance')

            data = Data([trace1, trace2])

            layout=Layout(
                        yaxis=YAxis(title='Explained variance in percent'),
                        title='Explained variance by different principal components')

            fig = Figure(data=data, layout=layout)
            py.iplot(fig)
```

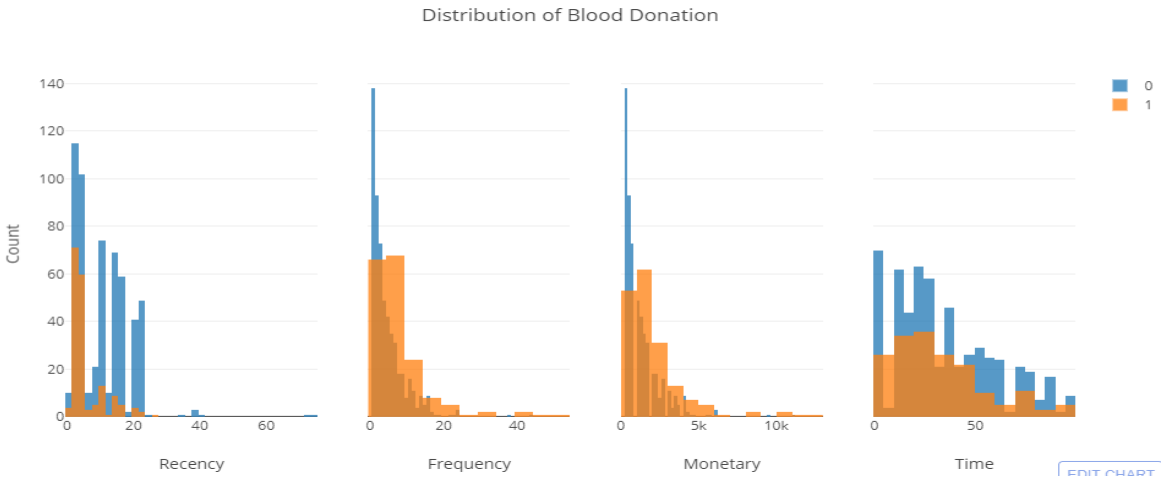
In [13]:
```
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1), eig_pairs[1][1].reshape(4,1)))

print('Matrix W:\n', matrix_w)
```

In [14]:
```
Y = X_std.dot(matrix_w)
```

In [15]:
```
traces = []

for name in (0,1):

    trace = Scatter(
        x=Y[y==name,0],
        y=Y[y==name,1],
        mode='markers',
        name=name,
        marker=Marker(
            size=12,
            line=Line(
                color='rgba(217, 217, 217, 0.14)',
                width=0.5),
            opacity=0.8))
    traces.append(trace)

data = Data(traces)
layout = Layout(showlegend=True,
                scene=Scene(xaxis=XAxis(title='PC1'),
                yaxis=YAxis(title='PC2'),))

fig = Figure(data=data, layout=layout)
py.iplot(fig)
```

OUTPUT :

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
0	2	50	12500	98	1
1	0	13	3250	28	1
2	1	16	4000	35	1
3	2	20	5000	45	1
4	1	24	6000	77	0

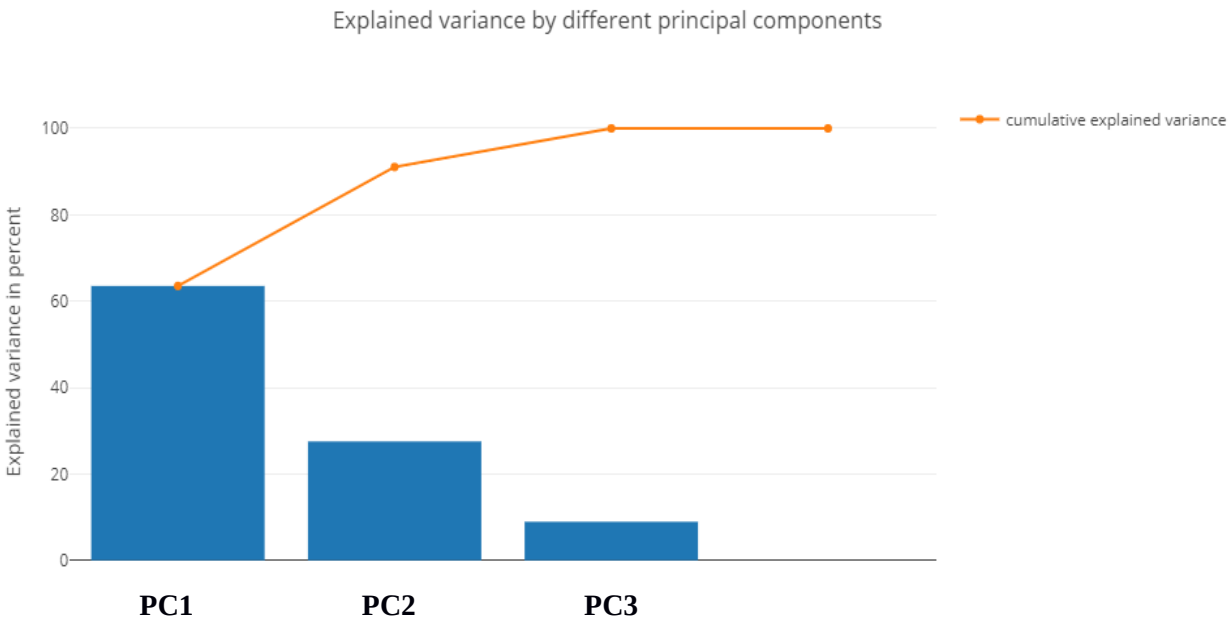


Covariance matrix
[[1.00133869 -0.18299011 -0.18299011 0.16083311]
[-0.18299011 1.00133869 1.00133869 0.63579026]
[-0.18299011 1.00133869 1.00133869 0.63579026]
[0.16083311 0.63579026 0.63579026 1.00133869]]

Eigenvectors
[[-9.34888912e-02 9.15456285e-01 -3.91407228e-01 -1.61303336e-19]
[6.11256218e-01 -8.51400658e-02 -3.45133316e-01 7.07106781e-01]
[6.11256218e-01 -8.51400658e-02 -3.45133316e-01 -7.07106781e-01]
[4.93954957e-01 3.83981939e-01 7.80106641e-01 1.41013311e-16]]

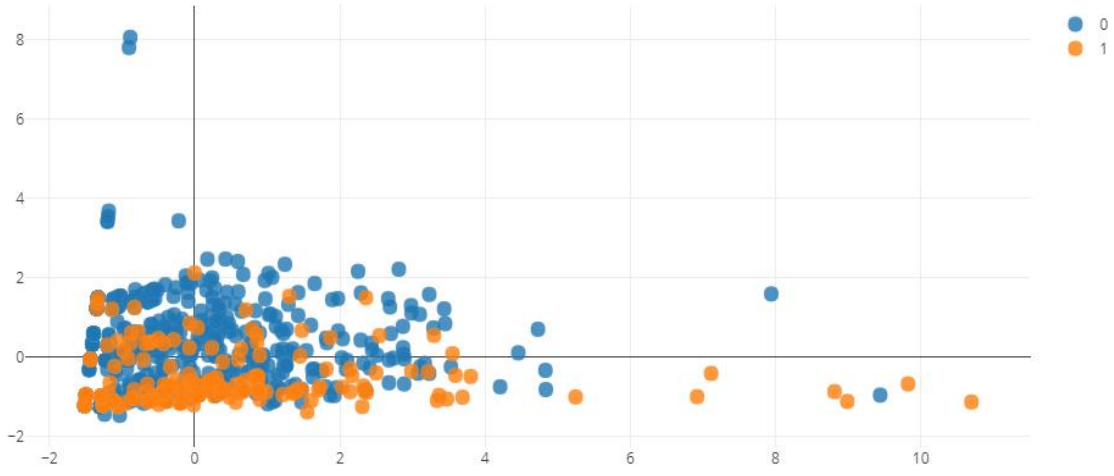
Eigenvalues
[2.54444576e+00 1.10283626e+00 3.58072735e-01 -1.38822770e-17]

Eigenvalues in descending order:
2.5444457574688304
1.102836259446772
0.35807273542710033
1.3882277025402316e-17
Matrix W:
[[-0.09348889 0.91545628]
[0.61125622 -0.08514007]
[0.61125622 -0.08514007]
[0.49395496 0.38398194]]



Above graph shows that if we use first two principal components then above 90% variance of our dataset will be explained.

PC1 vs PC2



Above plot is the scatter plot of our data points using first two principal components on axes
[0: did not donate blood, 1: donated the blood]

DISCUSSION :

A Multivariate Analysis problem could start out with a substantial number of correlated variables. Principal Component Analysis is a dimension-reduction tool that can be used advantageously in such situations.
Principal component analysis aims at reducing a large set of variables to a small set that still contains most of the information in the large set.

FINDING AND LEARNING :

From this experiment, we learn about implementing and understanding the concept of principle component analysis and dimensionality reduction.
The advantages of applying dimensionality reduction methods are:

1) Improved model interpretability.
2) Faster training time.
3) Reduction in overfitting the models.
4) Reduced noise in data.