Principal Component Analysis

- Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning.
- It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation.
- hese new transformed features are called the Principal Components.
- It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various communication channels. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

Some common terms used in PCA algorithm:

Dimensionality: It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.

Correlation: It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.

Orthogonal: It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.

Eigenvectors: If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.

Covariance Matrix: A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Steps for PCA algorithm

- Getting the dataset

Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.

- Representing data into a structure

Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

- Standardizing the data

In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.

If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

- Calculating the Covariance of Z

To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.

- Calculating the Eigen Values and Eigen Vectors

Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

- Sorting the Eigen Vectors

In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

- Calculating the new features Or Principal Components

Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

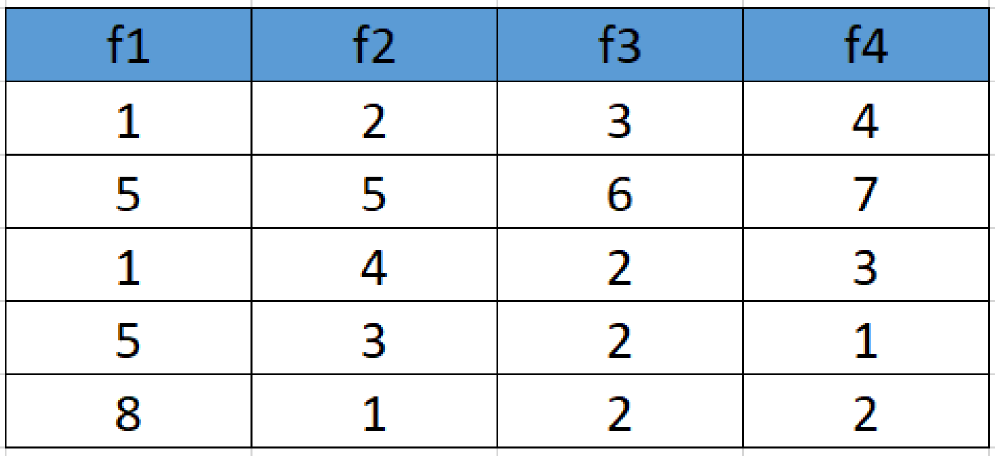
- Remove less or unimportant features from the new dataset.

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Mathematical Example

1. Standardize the Dataset

Assume we have the below dataset which has 4 features and a total of 5 training examples.

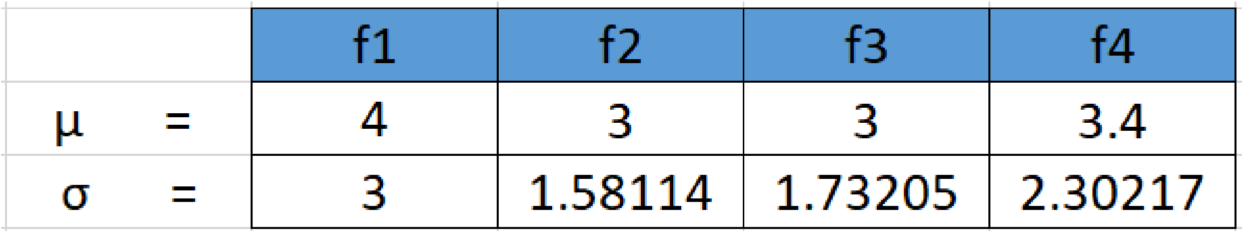


f1	f2	f3	f4
1	2	3	4
5	5	6	7
1	4	2	3
5	3	2	1
8	1	2	2

First, we need to standardize the dataset and for that, we need to calculate the mean and standard deviation for each feature.

$$x_{new} = \frac{x - \mu}{\sigma}$$

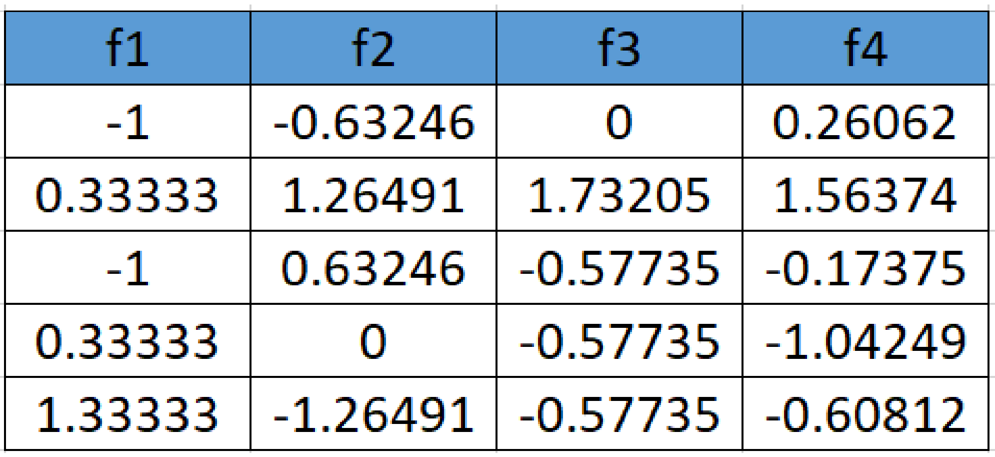
Standardization formula



	f1	f2	f3	f4
μ =	4	3	3	3.4
σ =	3	1.58114	1.73205	2.30217

Mean and standard deviation before standardization

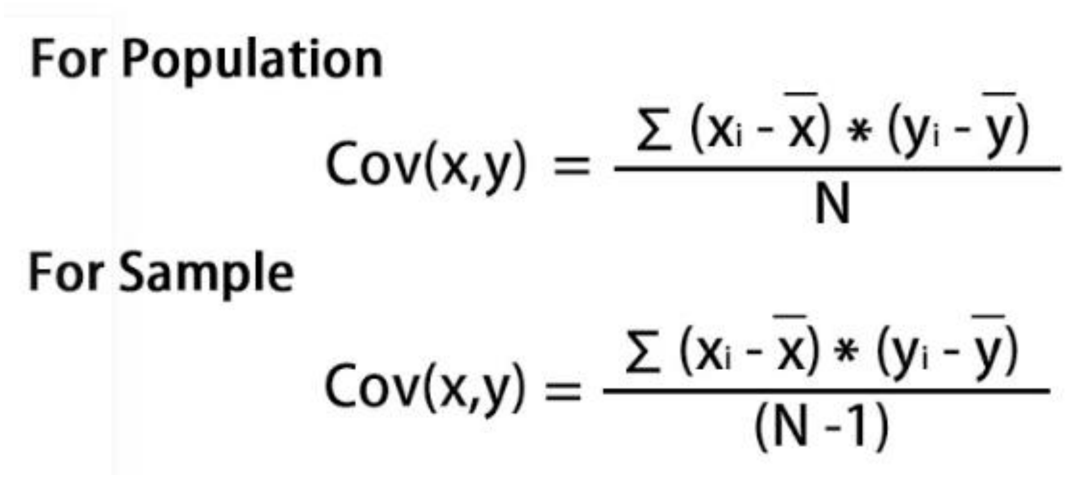
After applying the formula for each feature in the dataset is transformed as below:



f1	f2	f3	f4
-1	-0.63246	0	0.26062
0.33333	1.26491	1.73205	1.56374
-1	0.63246	-0.57735	-0.17375
0.33333	0	-0.57735	-1.04249
1.33333	-1.26491	-0.57735	-0.60812

2. Calculate the covariance matrix for the whole dataset

The formula to calculate the covariance matrix:



For Population

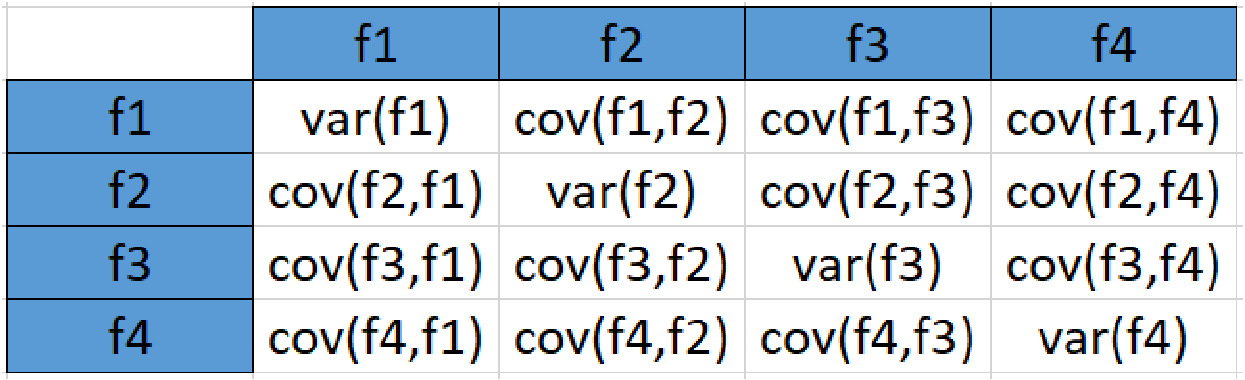
$$\mathrm{Cov}(x,y) = \frac{\Sigma\,(x_i - \overline{x}) * (y_i - \overline{y})}{N}$$

For Sample

$$\mathrm{Cov}(x,y) = \frac{\Sigma\,(x_i - \overline{x}) * (y_i - \overline{y})}{(N - 1)}$$

Covariance Formula

the covariance matrix for the given dataset will be calculated as below



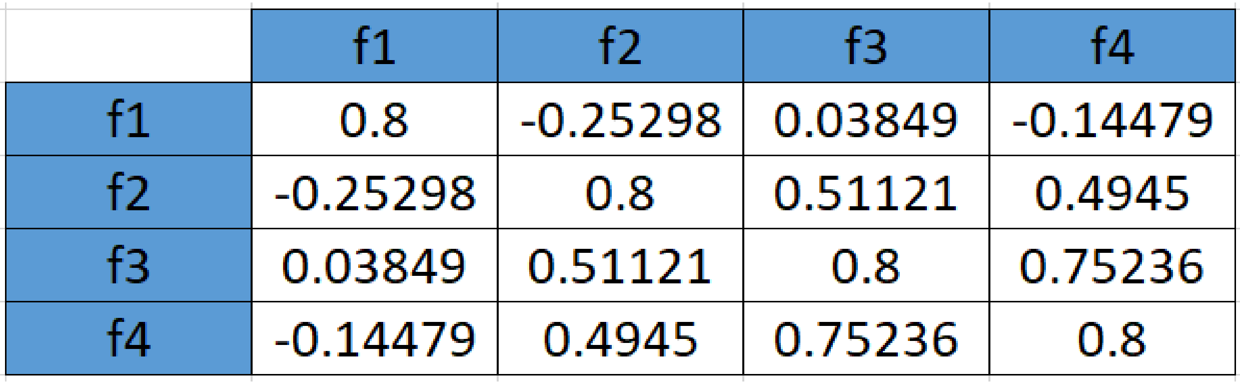
	f1	f2	f3	f4
f1	var(f1)	cov(f1,f2)	cov(f1,f3)	cov(f1,f4)
f2	cov(f2,f1)	var(f2)	cov(f2,f3)	cov(f2,f4)
f3	cov(f3,f1)	cov(f3,f2)	var(f3)	cov(f3,f4)
f4	cov(f4,f1)	cov(f4,f2)	cov(f4,f3)	var(f4)

Since we have standardized the dataset, so the **mean for each feature is 0** and the standard deviation is 1.

var(f1) = $((-1.0-0)^2 + (0.33-0)^2 + (-1.0-0)^2 + (0.33-0)^2 + (1.33-0)^2)/5$
var (f1) = 0.8

cov(f1,f2) =
((-1.0–0)*(-0.632456-0) +
(0.33–0)*(1.264911-0) +
(-1.0–0)* (0.632456-0)+
(0.33–0)*(0.000000 -0)+
(1.33–0)*(-1.264911–0))/5
cov(f1,f2 = -0.25298

In the similar way be can calculate the other covariances and which will result in the below covariance matrix



	f1	f2	f3	f4
f1	0.8	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8	0.51121	0.4945
f3	0.03849	0.51121	0.8	0.75236
f4	-0.14479	0.4945	0.75236	0.8

covariance matrix (population formula)

3. Calculate eigenvalues and eigen vectors.

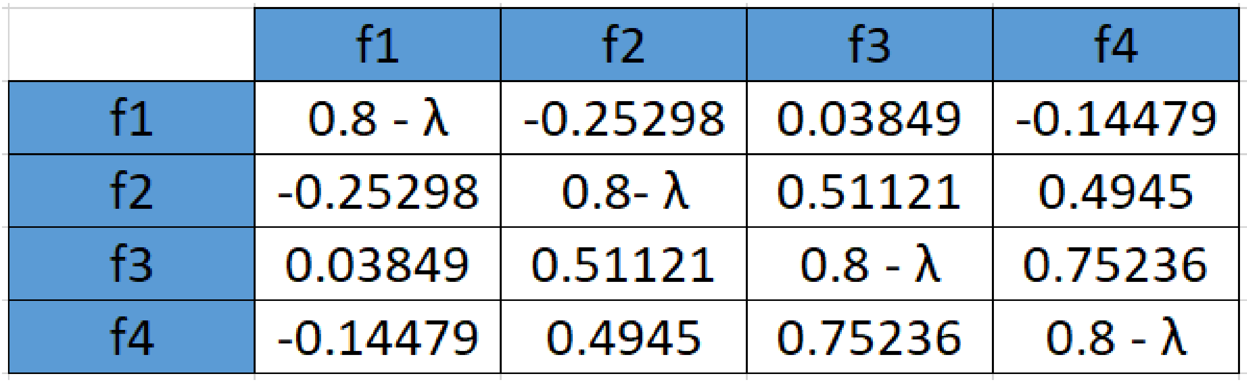
An **eigenvector** is a nonzero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding **eigenvalue** is the factor by which the eigenvector is scaled.

Let A be a square matrix (in our case the covariance matrix), ν a vector and λ a scalar that satisfies Aν = λν, then λ is called eigenvalue associated with eigenvector ν of A.
Rearranging the above equation,

$$A\nu - \lambda\nu = 0 \; ; \; (A - \lambda I)\nu = 0$$

Since we have already know v is a non- zero vector, only way this equation can be equal to zero, if

det(A-λI) = 0



	f1	f2	f3	f4
f1	0.8 - λ	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8- λ	0.51121	0.4945
f3	0.03849	0.51121	0.8 - λ	0.75236
f4	-0.14479	0.4945	0.75236	0.8 - λ

A-λI = 0

Solving the above equation = 0

λ = 2.51579324 , 1.0652885 , 0.39388704 , 0.02503121

Eigenvectors:

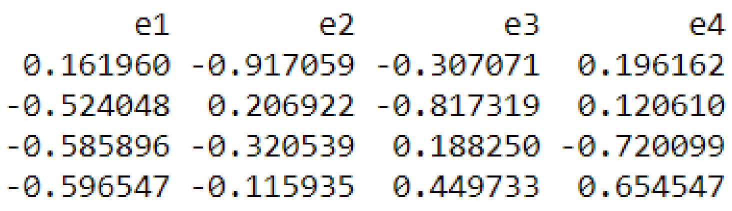
Solving the (A-λI)v = 0 equation for v vector with different λ values:

$$\begin{pmatrix} 0.800000-\lambda & -(0.252982) & 0.038490 & -(0.144791) \\ -(0.252982) & 0.800000-\lambda & 0.511208 & 0.494498 \\ 0.038490 & 0.511208 & 0.800000-\lambda & 0.752355 \\ -(0.144791) & 0.494498 & 0.752355 & 0.800000-\lambda \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = 0$$

For λ = *2.51579324, solving the above equation using Cramer's rule, the values for v vector are*

v1 = 0.16195986
v2 = -0.52404813
v3 = -0.58589647
v4 = -0.59654663

Going by the same approach, we can calculate the eigen vectors for the other eigen values. We can from a matrix using the eigen vectors.



```
        e1        e2        e3        e4
 0.161960 -0.917059 -0.307071  0.196162
-0.524048  0.206922 -0.817319  0.120610
-0.585896 -0.320539  0.188250 -0.720099
-0.596547 -0.115935  0.449733  0.654547
```

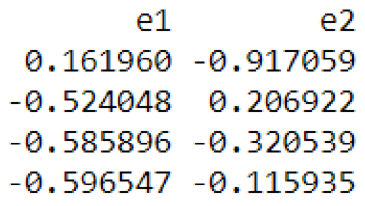
eigenvectors(4 * 4 matrix)

_4._ Sort eigenvalues and their corresponding eigenvectors.

Since eigenvalues are already sorted in this case so no need to sort them again.

_5._ Pick k eigenvalues and form a matrix of eigenvectors

If we choose the top 2 eigenvectors, the matrix will look like this:

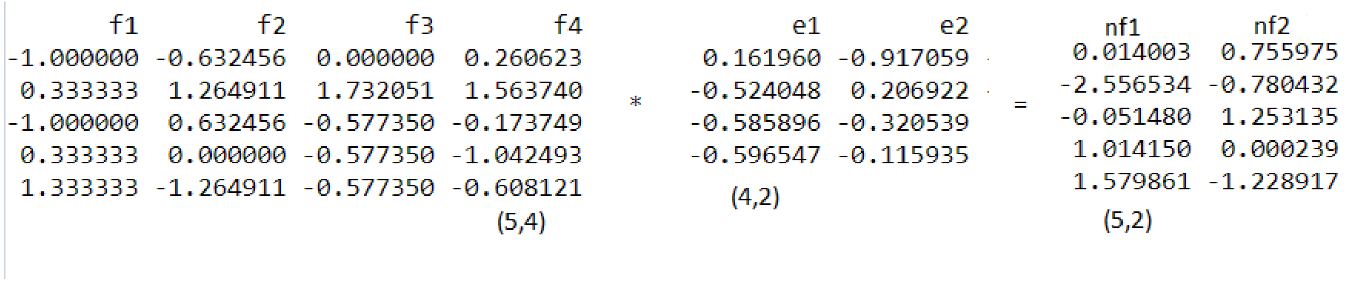


```
        e1        e2
 0.161960 -0.917059
-0.524048  0.206922
-0.585896 -0.320539
-0.596547 -0.115935
```

Top 2 eigenvectors(4*2 matrix)

6. Transform the original matrix.

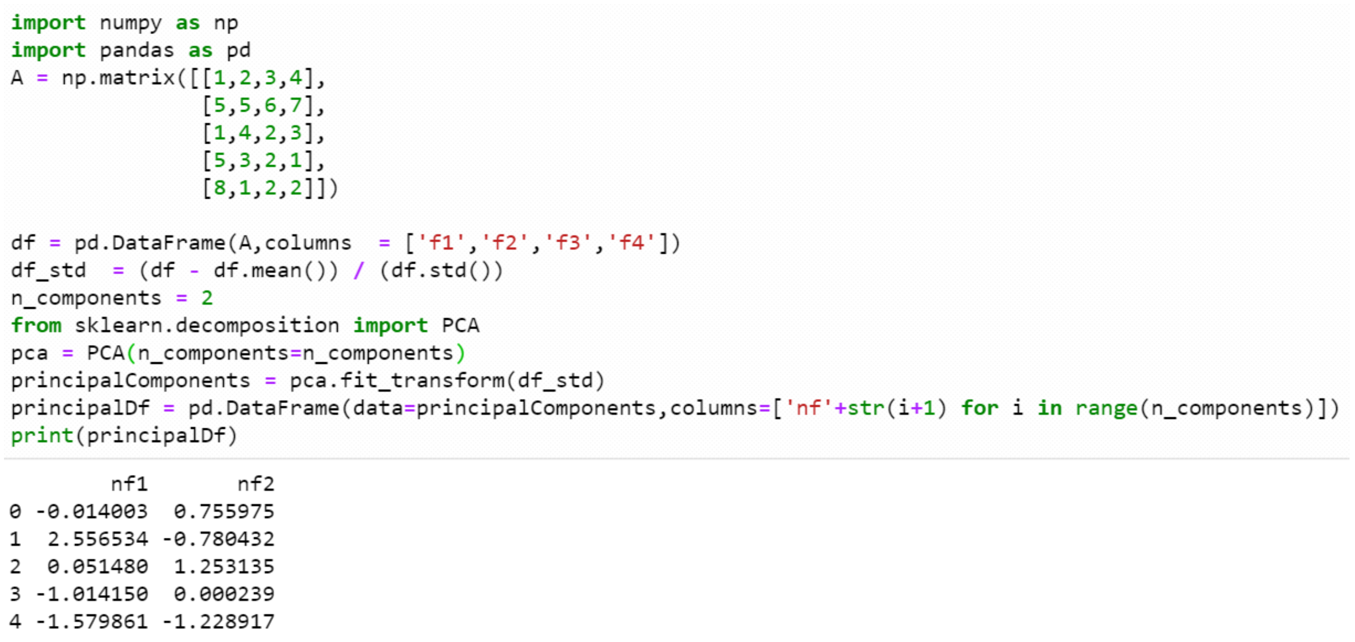
Feature matrix * top k eigenvectors = Transformed Data



```
       f1        f2        f3        f4             e1        e2          nf1       nf2
-1.000000 -0.632456  0.000000  0.260623       0.161960 -0.917059     0.014003  0.755975
 0.333333  1.264911  1.732051  1.563740   *  -0.524048  0.206922  = -2.556534 -0.780432
-1.000000  0.632456 -0.577350 -0.173749      -0.585896 -0.320539    -0.051480  1.253135
 0.333333  0.000000 -0.577350 -1.042493      -0.596547 -0.115935     1.014150  0.000239
 1.333333 -1.264911 -0.577350 -0.608121                              1.579861 -1.228917
                                 (5,4)          (4,2)                  (5,2)
```

Data Transformation

Python code with sklearn library



```python
import numpy as np
import pandas as pd
A = np.matrix([[1,2,3,4],
               [5,5,6,7],
               [1,4,2,3],
               [5,3,2,1],
               [8,1,2,2]])

df = pd.DataFrame(A,columns  = ['f1','f2','f3','f4'])
df_std  = (df - df.mean()) / (df.std())
n_components = 2
from sklearn.decomposition import PCA
pca = PCA(n_components=n_components)
principalComponents = pca.fit_transform(df_std)
principalDf = pd.DataFrame(data=principalComponents,columns=['nf'+str(i+1) for i in range(n_components)])
print(principalDf)
```

```
        nf1       nf2
0 -0.014003  0.755975
1  2.556534 -0.780432
2  0.051480  1.253135
3 -1.014150  0.000239
4 -1.579861 -1.228917
```

Ccode snippet for PCA using Sklearn

The results are the same, only change in the direction of the PC1, which according to me, doesn't make any difference as mentioned here also. So we have successfully converted our data from 4 dimensional to 2 dimensional. PCA is mostly useful when data features are highly correlated.