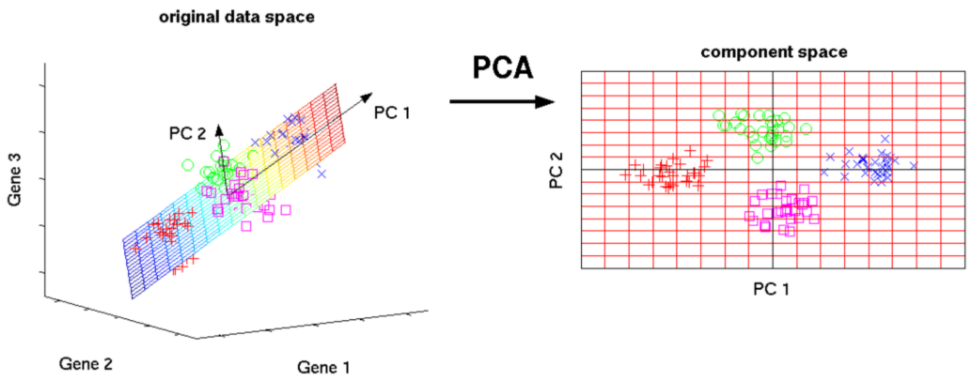
Principle Component Analysis - PCA

- PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
- PCA is a method of obtaining important variables (in form of components) from a large set of variables available in a data set.
- After applying PCA, we obtained fewer variables while minimising the loss of information, visualization also becomes much more meaningful.
- It is always performed on a symmetric correlation or covariance matrix. This means the matrix should be numeric and have standardized data.
- PCA determine less-important feature by using Eigen-Vector.

Eg - Let’s say we have a data set of dimension 300 (n) × 50 (p). n represents the number of observations and p represents number of predictors. Since we have a large p = 50, there can be p(p-1)/2 scatter plots i.e more than 1000 plots possible to analyze the variable relationship. Wouldn’t is be a tedious job to perform exploratory analysis on this data ?

In this case, it would be a clear approach to select a subset of p (p << 50) predictor which captures as much information.

The image below shows the transformation of a high dimensional data (3 dimension) to low dimensional data (2 dimension) using PCA. Not to forget, each resultant dimension is a linear combination of p features.



Approch to perform PCA -

Suppose we have an 'n' dimensional data and we want to reduce it to 'k' dimensions. We will do it in steps.

1. Data is mean normalised and feature scaled.
2. We find out the covariance matrix of our data set. Now we want to reduce the number of features i.e. dimensions. But cutting off features means loss of information. We want to minimise the loss of information i.e. we want to keep the maximum variance. So, we want to find out the directions in which variance is maximum.
3. We find out the Eigenvectors of the covariance matrix. As we have data in 'n' dimensions, we will find 'n' Eigenvectors corresponding to 'n' Eigenvalues.

4. We will select ‘k’ Eigenvectors corresponding to the ‘k’ largest Eigenvalues and will form a matrix in which each Eigenvector will constitute a column. We will call this matrix as U. Now it’s the time to find the reduced data points. Suppose you want to reduce a data point 'a' in the data set to 'k' dimensions. To do so, you have to just transpose the matrix U and multiply it with the vector 'a'. You will get the required vector in 'k' dimensions.

Steps to Perform PCA -

1. Standardize the dataset.
2. Calculate the covariance matrix for the features in the dataset.
3. Calculate the eigenvalues and eigenvectors for the covariance matrix.
4. Sort eigenvalues and their corresponding eigenvectors.
5. Pick k eigenvalues and form a matrix of eigenvectors.
6. Transform the original matrix.

Let's go to each step one by one.

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.

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

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

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

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

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

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), v a vector and λ a scalar that satisfies Av = λv, then λ is called eigenvalue associated with eigenvector v of A.
Rearranging the above equation, Av-λv =0
(A-λI)v = 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

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
```

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
```

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)
```

So we have successfully converted our data from 4 dimensional to 2 dimensional. PCA is mostly useful when data features are highly correlated.

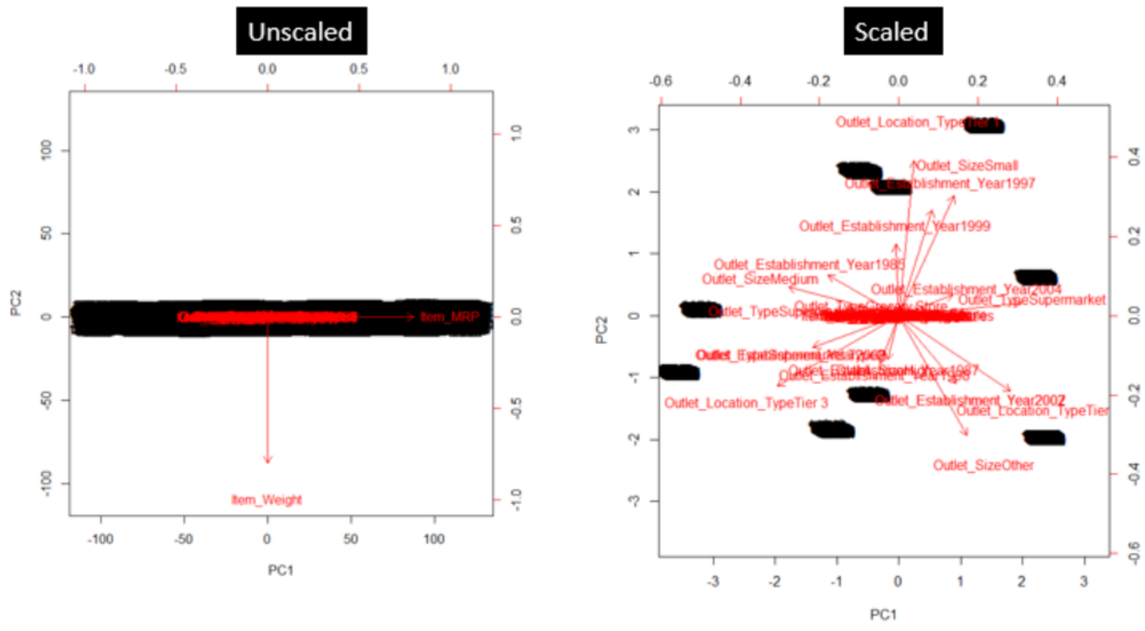
Compare with sklearn library

In [7]:
```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(
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
```

Why is normalization of variables necessary in PCA ?

The principal components are supplied with normalized version of original predictors. This is because, the original predictors may have different scales.
For example: Imagine a data set with variables measuring units as gallons, kilometers, light years etc. It is definite that the scale of variances in these variables will be large.



Points to Remember for PCA

1. PCA is used to overcome features redundancy in a data set.
2. These features are low dimensional in nature.
3. These features a.k.a components are a resultant of normalized linear combination of original predictor variables.
4. These components aim to capture as much information as possible with high explained variance.
5. The first component has the highest variance followed by second, third and so on.

6. The components must be uncorrelated (remember orthogonal direction ?).
7. Normalizing data becomes extremely important when the predictors are measured in different units.
8. PCA works best on data set having 3 or higher dimensions. Because, with higher dimensions, it becomes increasingly difficult to make interpretations from the resultant cloud of data.
9. PCA is applied on a data set with numeric variables.
10. PCA is a tool which helps to produce better visualizations of high dimensional data.

Refer - https://medium.com/analytics-vidhya/understanding-principle-component-analysis-pca-step-by-step-e7a4bb4031d9 (https://medium.com/analytics-vidhya/understanding-principle-component-analysis-pca-step-by-step-e7a4bb4031d9)