**PCA Simplified: Step-by-Step Understanding with NumPy.**

[[Rakesh M K](https://medium.com/@mkk.rakesh?source=post_page-----5aef313cd34f--------------------------------)](https://medium.com/@mkk.rakesh?source=post_page-----5aef313cd34f--------------------------------)

[[Artificial Intelligence in Plain English](https://ai.plainenglish.io/?source=post_page-----5aef313cd34f--------------------------------)](https://ai.plainenglish.io/?source=post_page-----5aef313cd34f--------------------------------)

[Rakesh M K](https://medium.com/@mkk.rakesh?source=post_page-----5aef313cd34f--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F5796472804b9&operation=register&redirect=https%3A%2F%2Fai.plainenglish.io%2Fpca-simplified-step-by-step-understanding-with-numpy-5aef313cd34f&user=Rakesh+M+K&userId=5796472804b9&source=post_page-5796472804b9----5aef313cd34f---------------------post_header-----------)

Published in

[Artificial Intelligence in Plain English](https://ai.plainenglish.io/?source=post_page-----5aef313cd34f--------------------------------)

·

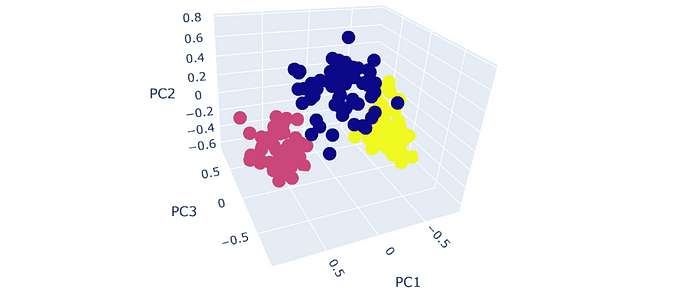
5 min read

·

Oct 18

87

1



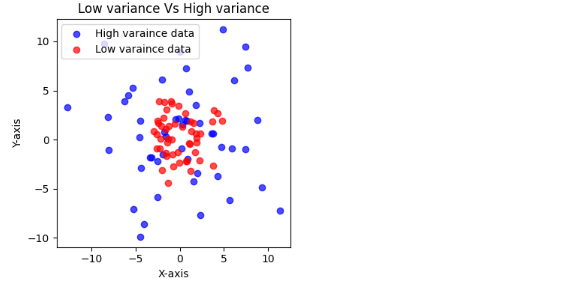
3 Principal components after performing PCA.

Every data scientist has likely encountered the challenge of dealing with higher dimensions and the need for **dimensionality reduction**. Quite often, we turn to **Principal Component Analysis** (PCA) as a powerful technique for this purpose. In this article, I intend to go through a step-by-step process of performing PCA with NumPy library. Also compare our results with those principal components obtained using scikit-learn library.

Before understanding the inner working of PCA, one should know the importance of **variance**indata science. PCA leverages the measures variance and covariance (variance among the features) within the data.

**Variance in simple words**

Variance can be viewed as a **measure of information content**. High variance indicates that data points are more spread out and carry more information. In PCA, variance is a key factor in determining the importance of each principal component. Principal components are ranked by the amount of variance they explain.

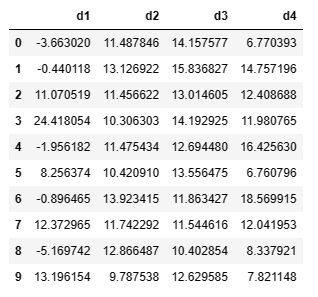


**How does PCA work?**

PCA makes use of the **covariance matrix** of the data to identify the principal components. It transforms the original features into a new set of**uncorrelated variables called principal components**, which are **orthogonal** to each other. This orthogonality is crucial because it means that the principal components do not contain redundant information and capture different aspects of the data’s variation.

To start with, four random normal sequence with different mean and standard deviation are generated. Sequences are converted to a pandas dataframe for better readability.

import numpy as np  
import pandas as pd  
  
d1 = np.random.normal(3,10,10)   
d2 = np.random.normal(12,1.5,10)   
d3 = np.random.normal(14,2,10)   
d4 = np.random.normal(11,3,10)   
data= pd.DataFrame({'d1':d1,'d2':d2,'d3':d3,'d4':d4})  
data



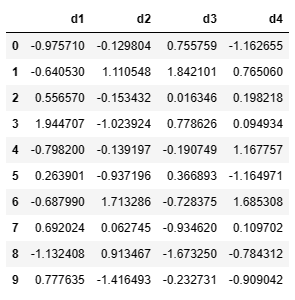
Generated data

**1. Standardization of data**

PCA demands standardized data to ensures that all variables have the same scale. This is important because PCA aims to **maximize the varianc**e along each principal component, and variables with larger scales could dominate the analysis.

The code snippet below will standardize the data.

for c in data.columns:  
 mean = data[c].mean() # mean of each column  
 std = data[c].std() # standard deviation of each column  
 data[c] = (data[c]-mean)/std # standardizing the data  
data



Standardized data

**2. Generate covariance matrix.**

The variance between every pair of features is calculated in this step. Covariance matrix of n features will be a square matrix of dimension n\*n.

'''Covariance matrix of data'''  
cov\_matrix = np.cov(data, rowvar=False)  
  
cov\_matrix  
array([[ 2.19128029e-31, -2.62953635e-31, -1.75302423e-31,  
 -3.50604847e-31],  
 [-2.62953635e-31, 2.18209243e+00, -4.19347099e-01,  
 -6.01908383e-01],  
 [-1.75302423e-31, -4.19347099e-01, 1.71062924e+00,  
 -3.95746952e+00],  
 [-3.50604847e-31, -6.01908383e-01, -3.95746952e+00,  
 1.48307344e+01]])

**3. Determine eigen values and eigen vectors of covariance matrix**

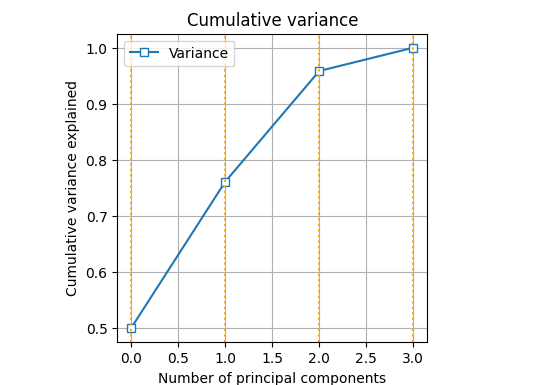
**Eigen values** of a square matrix provide information about how much variance exists in a particular direction. More precisely, they are the solutions to the characteristic equation of the covariance matrix. Each eigen value will have an **eigen vector**whichrepresent the principal components themselves. i.e., **the directions in feature space along which the data exhibits the most variance**.

So, once we sort the eigenvalues in descending order, we can find the variance explained by each eigen value in the same order. By calculating or plotting the cumulative variance, the decision in number of principal components needed can be taken. Generally, first n components which explains more than 90% variance is chosen.

'''Calculate eigen values and eigen vectors'''  
  
eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)  
eigenvalues\_sorted = sorted(eigenvalues,reverse=True) # sort eigenvalues in decsnding order  
eigenvectors\_sorted = np.sort(eigenvectors)[::-1] # sort eigenvectors in descending order  
  
print(f' Sorted eigen values:\n\n {eigenvalues\_sorted} \n\n Sorted eigen vectors:\n\n{eigenvectors\_sorted}')[2.0039618890416384,  
   
 Sorted eigen values:  
  
 [2.0039618890416384, 1.0416399653392925, 0.7884543845041312, 0.16594376111493936]   
  
 Sorted eigen vectors:  
  
[[-0.46048295 0.43651653 0.53482711 0.55800424]  
 [-0.54785206 -0.07623506 0.2024481 0.80812196]  
 [-0.73651462 -0.67146292 -0.04399548 0.06890685]  
 [-0.5110653 0.2369614 0.54415785 0.6217345 ]]

Now calculate the variance explained by each eigen values and plot the cumulative variance.

'''Varance expalined by eigen values'''  
  
var\_explained = [eigen\_value/sum(eigenvalues) for eigen\_value in eigenvalues\_sorted]  
cumularive\_variance = np.cumsum(var\_explained) #calculate cumulative variance  
  
'''Plot cumulative variance'''  
plt.subplots(figsize=(4,4))  
plt.plot(cumularive\_variance,marker='s',markerfacecolor='w')  
plt.grid()  
for i in range(len(var\_explained)):  
 plt.axvline(x=i,linestyle=':',color='orange')  
plt.title('Cumulative variance')   
plt.xlabel('Number of principal components')  
plt.ylabel('Cumulative variance explained')  
plt.show()



It can be observed from the above plot that first 3 principal components explain more 95% of the variance. So, choosing first 3 components will reduce the dimension of data by 1 which only costs less than 5% of variance.

**4. Calculate of Principal components**

Selecting 3 as number of principal components, the **dot product**of original data and first 3 eigen vectors (sorted in descending order) yields the principal components.

first3\_eigenvectors = eigenvectors[:,:3]  
principal\_omponents = np.dot(data,first3\_eigenvectors)  
print(f'first 3 principal componenets corresponding to data:\n\n{principal\_omponents}')  
  
first 3 principal componenets corresponding to data:  
  
[[ 0.24460329 0.02912104 -1.66373175]  
 [-1.07360878 -0.29705389 -1.0293904 ]  
 [ 0.3179193 -0.0861592 0.45444053]  
 [ 1.85967002 -0.2576559 0.88054091]  
 [-0.91723045 1.03473961 0.26597174]  
 [ 1.38362212 0.01888867 -0.64575351]  
 [-2.44829619 -0.11905992 0.83632675]  
 [ 0.09471117 -0.28074432 1.00074264]  
 [-1.20715194 -0.30885235 -0.26520179]  
 [ 1.74576146 0.26677627 0.16605488]]

Since we obtained the components through all processes, let’s generate the same using **scikit-learn library and cross check** with what we got.

from sklearn.decomposition import PCA  
pca = PCA(n\_components=3) # make an instance of PCA with 3 principle components  
principal\_components = pca.fit\_transform(data) #fit the data to PCA and get components  
print(f'first 3 principal components using scikit-learn:\n\n{principal\_components}')  
  
first 3 principal components using scikit-learn:  
  
[[-0.24460329 0.2512241 1.66373175]  
 [ 1.07360878 -1.82256036 1.0293904 ]  
 [-0.3179193 -0.24053566 -0.45444053]  
 [-1.85967002 -1.07026284 -0.88054091]  
 [ 0.91723045 -0.27166571 -0.26597174]  
 [-1.38362212 0.32860849 0.64575351]  
 [ 2.44829619 -0.26776272 -0.83632675]  
 [-0.09471117 0.52830881 -1.00074264]  
 [ 1.20715194 1.97705412 0.26520179]  
 [-1.74576146 0.58759177 -0.16605488]]

We have got same numbers with **opposite sign.** This sign difference doesn’t affect the information it captures. It implies that only the orientation of the orthogonal components is different.

Theoretically, the principal components should be uncorrelated. It can be checked as below.

correlation\_matrix = np.corrcoef(principal\_components, rowvar=False)  
print(f'correlation\_matrix:\n\n {correlation\_matrix}\n\n correlation matrix rounded to 15 decimals:{np.round(correlation\_matrix,15)}')  
   
correlation\_matrix:  
  
 [[ 1.00000000e+00 1.36931910e-16 -2.08585238e-16]  
 [ 1.36931910e-16 1.00000000e+00 -1.23662396e-16]  
 [-2.08585238e-16 -1.23662396e-16 1.00000000e+00]]  
  
 correlation matrix rounded to 15 decimals:  
  
[[ 1. 0. -0.]  
 [ 0. 1. -0.]  
 [-0. -0. 1.]]

We could obtain three principal components which are orthogonal to each other or uncorrelated which is clear from the above block. Also, the dimension of data also has reduced by one...!

Hope this reading helped to get an idea how PCA functions work internally…*bests.*

**In Plain English**

*Thank you for being a part of our community! Before you go:*

* *Be sure to****clap****and****follow****the writer! 👏*
* *You can find even more content at*[***PlainEnglish.io***](https://plainenglish.io/)***🚀***
* *Sign up for our*[***free weekly newsletter***](http://newsletter.plainenglish.io/)*. 🗞️*
* *Follow us:*[***Twitter***](https://twitter.com/inPlainEngHQ)***(X***), [***LinkedIn***](https://www.linkedin.com/company/inplainenglish/), [***YouTube***](https://www.youtube.com/channel/UCtipWUghju290NWcn8jhyAw), [***Discord***](https://discord.gg/in-plain-english-709094664682340443)***.***
* *Check out our other platforms:****[Stackademic](https://stackademic.com/" \t "_blank)***, ***[CoFeed](https://cofeed.app/" \t "_blank)***, [***Venture***](https://venturemagazine.net/).