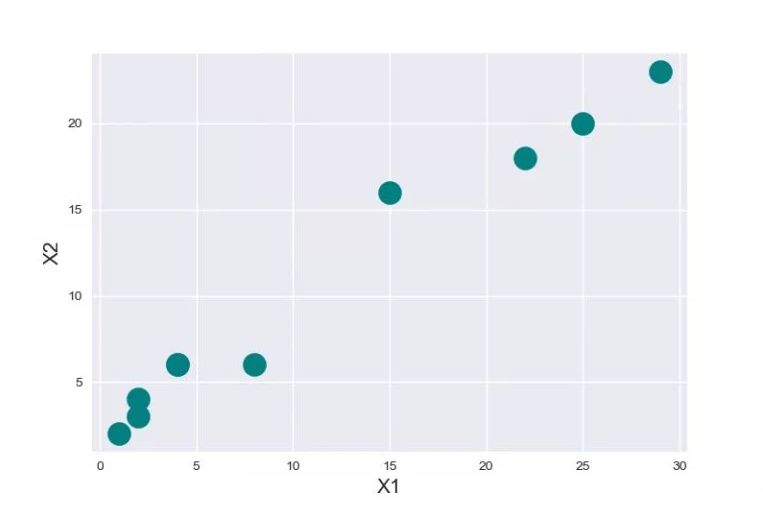
Principal Component Analysis in Python

Principal component analysis or PCA in short is famously known as a dimensionality reduction technique.

**Suppose we have a dataset** having two variables and 10 number of data points. If we were asked to visualize the data points, we can do it very easily. The result is very interpretable as well.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X1 | 2 | 8 | 1 | 4 | 22 | 15 | 25 | 29 | 4 | 2 |
| X2 | 3 | 6 | 2 | 6 | 18 | 16 | 20 | 23 | 6 | 4 |

Example Data points

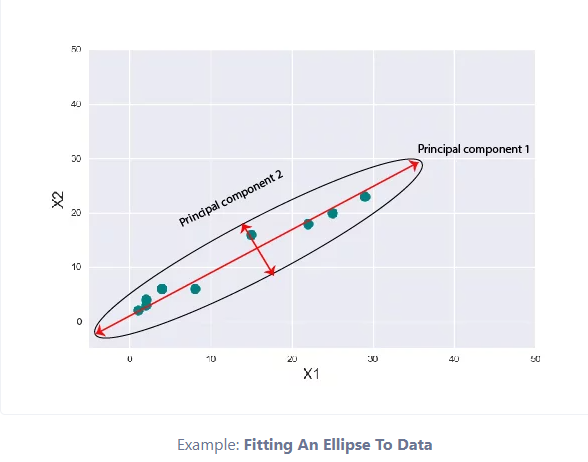


**Plotting Data On Two Dimensions**

Now if we try to increase the number of variables it gets almost impossible for us to imagine a dimension higher than three-dimensions.

Principal Component analysis reduces high dimensional data to lower dimensions while capturing maximum variability of the dataset. Data visualization is the most common application of PCA. PCA is also used to make the training of an algorithm faster by reducing the number of dimensions of the data.

Implementation of PCA with python

We can think of Principal Component analysis to be like fitting an n-dimensional ellipsoid to the data so that each axis of the ellipsoid represents a principal component. The larger the principal component axis the larger the variability in data it represents. ****

 Steps to implement PCA in Python

|  |
| --- |
| #Importing required libraries  import numpy as np |

1. Subtract the mean of each variable

Subtract the mean of each variable from the dataset so that the dataset should be centered on the origin. Doing this proves to be very helpful when calculating the covariance matrix.

|  |
| --- |
| #Generate a dummy dataset.  X = np.random.randint(10,50,100).reshape(20,5)  # mean Centering the data  X\_meaned = X - np.mean(X , axis = 0) |

Data generated by the above code have dimensions (20,5) i.e. 20 examples and 5 variables for each example. we calculated the mean of each variable and subtracted that from every row of the respective column.

2. Calculate the Covariance Matrix

Calculate the Covariance Matrix of the mean-centered data. You can know more about the covariance matrix in this really informative Wikipedia article [here](https://en.wikipedia.org/wiki/Covariance_matrix).

The covariance matrix is a square matrix denoting the covariance of the elements with each other. The covariance of an element with itself is nothing but just its Variance.

That’s why the diagonal elements of a covariance matrix are just the variance of the elements.

|  |
| --- |
| # calculating the covariance matrix of the mean-centered data.  cov\_mat = np.cov(X\_meaned , rowvar = False) |

We can find easily calculate covariance Matrix using numpy.cov( ) method. The default value for rowvar is set to True, remember to set it to False to get the covariance matrix in the required dimensions.

3. Compute the Eigenvalues and Eigenvectors

Now, compute the Eigenvalues and Eigenvectors for the calculated Covariance matrix. The Eigenvectors of the Covariance matrix we get are Orthogonal to each other and each vector represents a principal axis.

A Higher Eigenvalue corresponds to a higher variability. Hence the principal axis with the higher Eigenvalue will be an axis capturing higher variability in the data.

Orthogonal means the vectors are mutually perpendicular to each other. Eigenvalues and vectors seem to be very scary until we get the idea and concepts behind it.

|  |
| --- |
| #Calculating Eigenvalues and Eigenvectors of the covariance matrix  eigen\_values , eigen\_vectors = np.linalg.eigh(cov\_mat) |

NumPy linalg.eigh( ) method returns the eigenvalues and eigenvectors of a complex Hermitian or a real symmetric matrix.

4. Sort Eigenvalues in descending order

Sort the Eigenvalues in the descending order along with their corresponding Eigenvector.

Remember each column in the Eigen vector-matrix corresponds to a principal component, so arranging them in descending order of their Eigenvalue will automatically arrange the principal component in descending order of their variability.

Hence the first column in our rearranged Eigen vector-matrix will be a principal component that captures the highest variability.

|  |
| --- |
| #sort the eigenvalues in descending order  sorted\_index = np.argsort(eigen\_values)[::-1]    sorted\_eigenvalue = eigen\_values[sorted\_index]  #similarly sort the eigenvectors  sorted\_eigenvectors = eigen\_vectors[:,sorted\_index] |

np.argsort returns an array of indices of the same shape.

5. Select a subset from the rearranged Eigenvalue matrix

Select a subset from the rearranged Eigenvalue matrix as per our need i.e. number\_comp = 2. This means we selected the first two principal components.

|  |
| --- |
| # select the first n eigenvectors, n is desired dimension  # of our final reduced data.    n\_components = 2 #you can select any number of components.  eigenvector\_subset = sorted\_eigenvectors[:,0:n\_components] |

n\_components = 2 means our final data should be reduced to just 2 variables. if we change it to 3 then we get our data reduced to 3 variables.

6. Transform the data

Finally, transform the data by having a dot product between the Transpose of the Eigenvector subset and the Transpose of the mean-centered data. By transposing the outcome of the dot product, the result we get is the data reduced to lower dimensions from higher dimensions.

|  |
| --- |
| #Transform the data  X\_reduced = np.dot(eigenvector\_subset.transpose(),X\_meaned.transpose()).transpose() |

The final dimensions of X\_reduced will be ( 20, 2 ) and originally the data was of higher dimensions ( 20, 5 ).

Now we can visualize our data with the available tools we have. Hurray! Mission accomplished.

Complete Code for Principal Component Analysis in Python

Now, let’s just combine everything above by making a function and try our Principal Component analysis from scratch on an example.

|  |
| --- |
| import numpy as np    def PCA(X , num\_components):        #Step-1      X\_meaned = X - np.mean(X , axis = 0)        #Step-2      cov\_mat = np.cov(X\_meaned , rowvar = False)        #Step-3      eigen\_values , eigen\_vectors = np.linalg.eigh(cov\_mat)        #Step-4      sorted\_index = np.argsort(eigen\_values)[::-1]      sorted\_eigenvalue = eigen\_values[sorted\_index]      sorted\_eigenvectors = eigen\_vectors[:,sorted\_index]        #Step-5      eigenvector\_subset = sorted\_eigenvectors[:,0:num\_components]        #Step-6      X\_reduced = np.dot(eigenvector\_subset.transpose() , X\_meaned.transpose() ).transpose()        return X\_reduced |

We defined a function named PCA accepting data matrix and the number of components as input arguments.

We’ll use [IRIS dataset](https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data) and apply our PCA function to it.

|  |
| --- |
| import pandas as pd    #Get the IRIS dataset  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"  data = pd.read\_csv(url, names=['sepal length','sepal width','petal length','petal width','target'])    #prepare the data  x = data.iloc[:,0:4]    #prepare the target  target = data.iloc[:,4]    #Applying it to PCA function  mat\_reduced = PCA(x , 2)    #Creating a Pandas DataFrame of reduced Dataset  principal\_df = pd.DataFrame(mat\_reduced , columns = ['PC1','PC2'])    #Concat it with target variable to create a complete Dataset  principal\_df = pd.concat([principal\_df , pd.DataFrame(target)] , axis = 1) |

we should [standardize data](https://www.askpython.com/python/examples/standardize-data-in-python) wherever necessary before applying any ML algorithm to it. In the above code, we did not standardize our data, but we did so while implementing PCA.

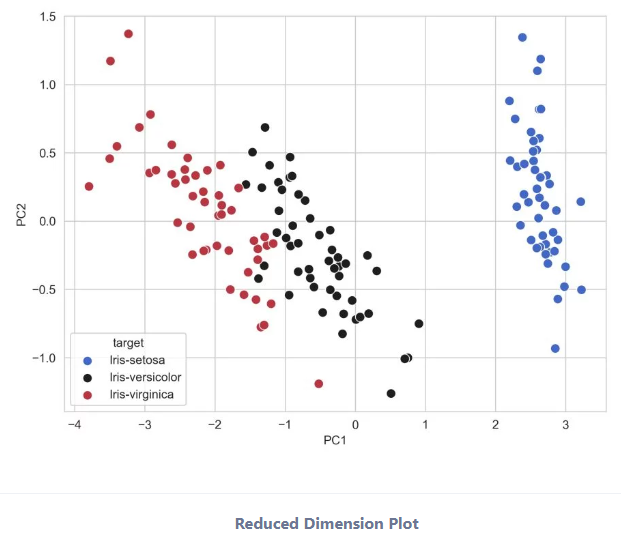
Let’s plot our results using the [seaborn](https://www.askpython.com/python-modules/python-seaborn-tutorial) and [matplotlib](https://www.askpython.com/python-modules/matplotlib/python-matplotlib) libraries.

import seaborn as sb

import matplotlib.pyplot as plt

plt.figure(figsize = (6,6))

sb.scatterplot(data = principal\_df , x = 'PC1',y = 'PC2' , hue = 'target' , s = 60 , palette= 'icefire')

****