How to manually calculate PCA :

In Data Science and Machine Learning, Principal Component Analysis (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.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

PCA finds directions of maximal variance of data. It finds directions that are mutually orthogonal. Mutually orthogonal means it's a global algorithm. Global means, that all the directions, all the new features that they find have a big global constraint, namely that they must be mutually orthogonal.

Let’s see how can we manually compute PCA given some random table of values (see the illustration) :

Step 1: Standardize the dataset

Step 2: Calculate the covariance matrix for the features in the dataset

Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix

Step 4: Sort eigenvalues and their corresponding eigenvectors

Step 5: Calculate eigenvector for each eigenvalue using Cramer’s rule

Step 6: Build eigenvectors matrix

Step 7: Pick k eigenvalues and form a matrix of eigenvectors

Step 8: Transform the original matrix

Principle Component Analysis (PCA) vs Linear Discriminant Analysis (LDA)

In Data Science and Machine Learning, PCA is an unsupervised dimensionality reduction technique, that ignores the class label.

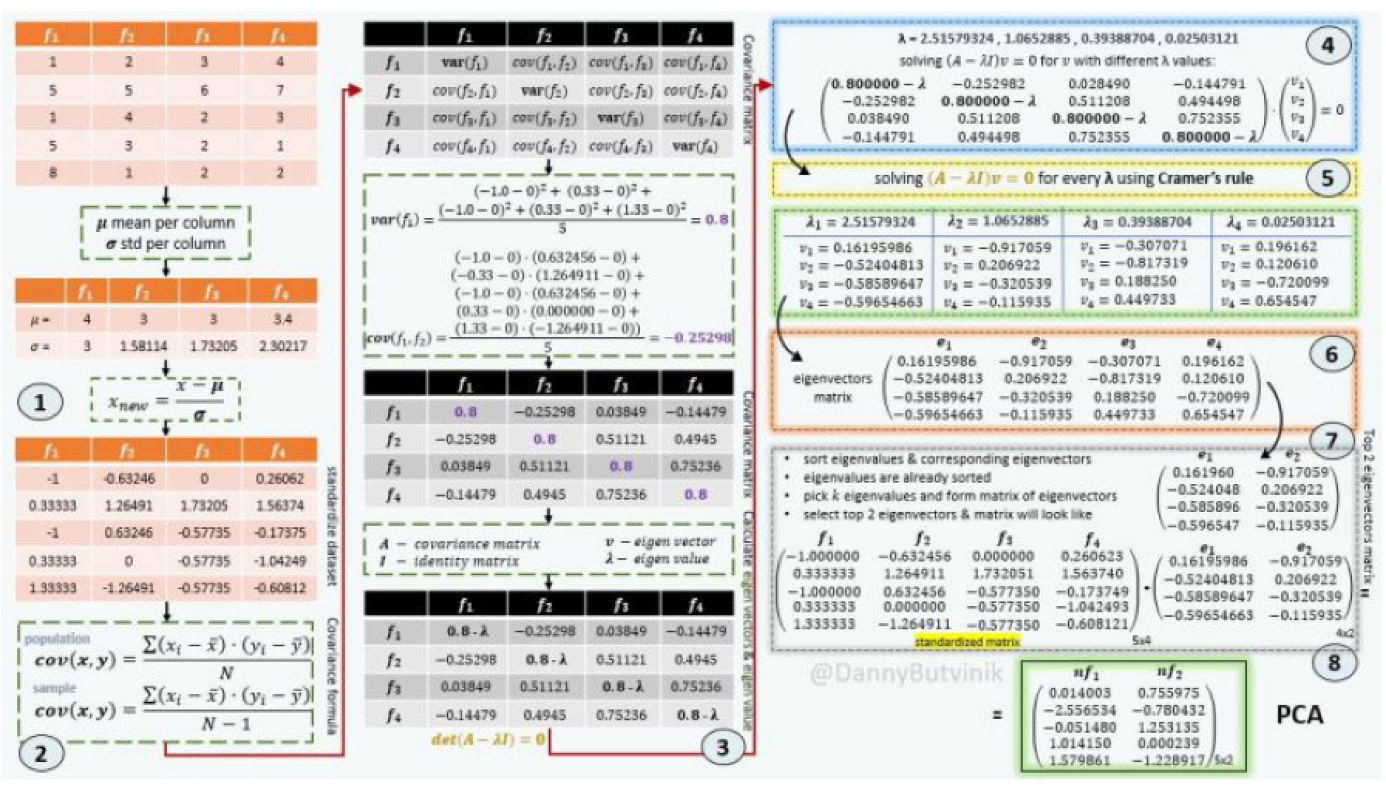
PCA focuses on capturing the direction of maximum variation in the data set.

LDA is a supervised dimensionality reduction method, that focuses on finding a feature subspace that maximizes the separability between the groups.

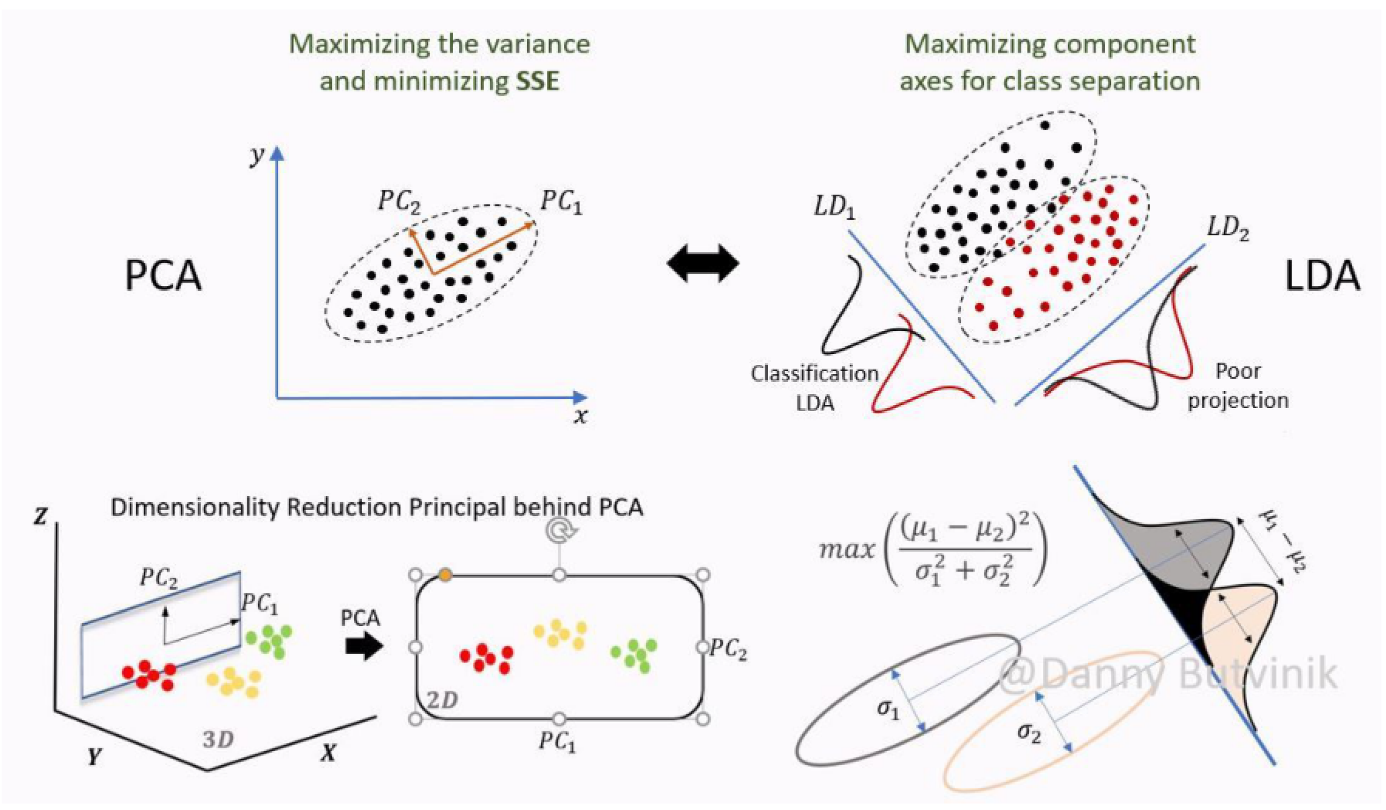
- PCA performs better in cases where a number of samples per class is less.
- LDA works better with large datasets having multiple classes; class separability is an important factor while reducing the dimensionality

In [1]:
```
import matplotlib.image as mpimg
```

In [2]:
```
img = mpimg.imread('C:\\Users\\u\\Desktop\\Analysis data\\PCA_Image.jpg')
plt.figure(figsize=(16,10))
plt.axis('off')
plt.imshow(img)
plt.show()
```



In [3]:
```
img = mpimg.imread('C:\\Users\\u\\Desktop\\Analysis data\\PCA_vs_LDA.jpg')
plt.figure(figsize=(16,10))
plt.axis('off')
plt.imshow(img)
plt.show()
```



Practical Implementation of Principal Component Analysis(PCA)

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eig
from numpy.linalg import eig
```

In [5]:
```
# 2-D Data
data = np.array([[3,4],[2,8],[6,9],[10,12]])
```

In [6]:
```
data
```

Out[6]:
```
array([[ 3,  4],
       [ 2,  8],
       [ 6,  9],
       [10, 12]])
```

In [7]:
```
#Create DataFrame

df = pd.DataFrame(data, columns = ['ML','DL'])
```

In [8]:
```
df
```

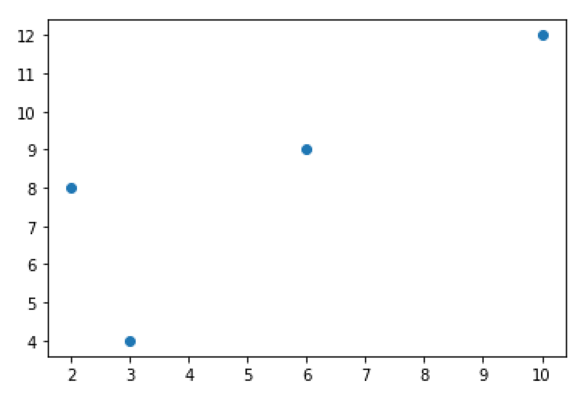
Out[8]:

	ML	DL
0	3	4
1	2	8
2	6	9
3	10	12

In [9]:
```
#Plot it on Scater plot

plt.scatter(df['ML'], df['DL'])
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x132dd97f580>
```



PCA STEPS :

In [10]:
```
'''1).Standarization of the data (Zero centric)
   2).COV matrix
   3).Eigen value and Eigen vector
   4).Find Principal Component'''
```

Out[10]:
```
'1).Standarization of the data (Zero centric)\n   2).COV matrix\n   3).Eigen value and Eigen vector\n   4).Find Principal Component'
```

In [11]:
```
data
```

Out[11]:
```
array([[ 3,  4],
       [ 2,  8],
       [ 6,  9],
       [10, 12]])
```

In [12]:
```
#Transpose the data

data.T
```

Out[12]:
```
array([[ 3,  2,  6, 10],
       [ 4,  8,  9, 12]])
```

In [13]:
```
#Find the mean

np.mean(data)
```

Out[13]:
```
6.75
```

In [14]:
```
#Mean Row wise
np.mean(data, axis = 1)
```

Out[14]:
```
array([ 3.5,  5. ,  7.5, 11. ])
```

In [15]:
```
#Mean Column wise
mean_by_col = np.mean(data.T, axis = 1)
```

STEP_01 - Standard Normal Distribution

In [16]:
```
scaled_data = data - mean_by_col
```

In [17]:
```
scaled_data
```

Out[17]:
```
array([[-2.25, -4.25],
       [-3.25, -0.25],
       [ 0.75,  0.75],
       [ 4.75,  3.75]])
```

In [18]:
```
#Not doing division by sigma as (sigma = 1) and data set too small
```

STEP_02 - COV matrix

In [19]:
```
#Relation b/W variable

cov_mat = np.cov(scaled_data.T)
```

STEP_03 - Eigen value and Eigen vector

In [20]:
```
eig_value,eig_vector = np.linalg.eig(cov_mat)
```

In [21]:
```
eig_value
```

Out[21]:
```
array([21.55203266,  2.28130068])
```

In [22]:
```
eig_vector
```

Out[22]:
```
array([[ 0.74289445, -0.66940857],
       [ 0.66940857,  0.74289445]])
```

STEP_04 - Principal Component

In [23]:
```
#Transpose the data

eig_vector.T.dot(scaled_data.T).T
```

Out[23]:
```
array([[-4.51649894, -1.65113213],
       [-2.58175911,  1.98985424],
       [ 1.05922727,  0.05511441],
       [ 6.03903078, -0.39383652]])
```

In [24]:
```
#Import PCA from sklearn.decomposition
from sklearn.decomposition import PCA
pca=PCA()
```

In [25]:
```
#Transform the scaled data

pca.fit_transform(scaled_data)
```

Out[25]:
```
array([[-4.51649894, -1.65113213],
       [-2.58175911,  1.98985424],
       [ 1.05922727,  0.05511441],
       [ 6.03903078, -0.39383652]])
```

In [26]:
```
    #Create DataFrame for scaled data

pd.DataFrame(data = pca.fit_transform(scaled_data), columns = ['PC1','PC2'])
```

Out[26]:

	PC1	PC2
0	-4.516499	-1.651132
1	-2.581759	1.989854
2	1.059227	0.055114
3	6.039031	-0.393837

In [27]:
```
#Inverse The Data

pca.inverse_transform(pca.fit_transform(scaled_data))
```

Out[27]:
```
array([[-2.25, -4.25],
       [-3.25, -0.25],
       [ 0.75,  0.75],
       [ 4.75,  3.75]])
```

In [28]:
```
scaled_data
```

Out[28]:
```
array([[-2.25, -4.25],
       [-3.25, -0.25],
       [ 0.75,  0.75],
       [ 4.75,  3.75]])
```

In [29]:
```
#Variance ratio b/w PC1 and PC2

pca.explained_variance_ratio_
```

Out[29]:
```
array([0.90428109, 0.09571891])
```

In [30]:
```
0.90428109+0.09571891
```

Out[30]:
```
1.0
```

We have to select PC1 as it covers 90% of dataset variation

In [ ]: