1. **DIMENSION REDUCTION TECHNIQUE: PRINCIPAL COMPONENT ANALYSIS:-**

## **IMPORTANT TERMINOLOGY FOR UNDERSTANDING PCA: MATRIX MULTIPLICATION:-**

**To multiply an** **m×n** **matrix by an** **n×p** **matrix,** the **n**s **must be the same**,   
**and the result is an** **m×p** **matrix.**

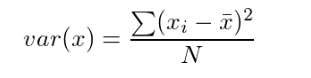
matrix multiply rows cols

* **The number of** **columns of the 1st matrix** **must equal the number** of **rows of the 2nd matrix**.
* **The result will have the same number of** **rows as the 1st matrix**, **and the same number of** **columns as the 2nd matrix**.

## **IMPORTANT TERMINOLOGY FOR UNDERSTANDING PCA: VARIANCE:-**

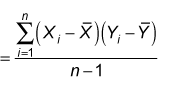
* **This is measure of variability or how spread the data is.**
* **Mathematically this is averaged squared from mean score as shown below:-**

**Var(x)=**



## **IMPORTANT TERMINOLOGY FOR UNDERSTANDING PCA: COVARAIANCE:-**

* This defined as **how interrelated two data sets are**
* This is **defined as measure of extent in which corresponding elements from two sets of ordered data move in same direction or how two variables change together.**
* Denoted **by cov(x,y) :- Covariance of x and y**



***Xi* is the value of X in *ith*dimension*.*  
*X bar* and *Y bar* denote the corresponding mean values of X and Y**

**N= number of samples. We are doing n-1 because of bias correction.**



**e.g. X=2.1,2.5,3.6,4.0**

**Y=8,10,12,14**

**Mean of x (x bar)= 2.1+2.5+3.6+4/4 = 3.05**

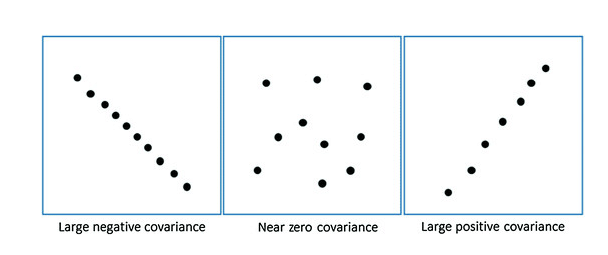
**Mean of y(y bar)=8+10+12+14/4 =11**

**Cov(X,Y)= (2.1-3.05)(8-11)+(2.5-3.05)(10-11)+(3.6-3.05)(12-11)+(4-3.05)(14-11)**

**=-(3.95-1.55+0.55+2.85)/3**

**=1.933**

* **From plot below:-**



* **Positive covariance means x and y datasets are positively correlated** i.e. if x increases then y increases or of y decreases then x decreases
* **Negative covariance means x and y datasets are negatively correlated** i.e. if x decreases then y increases and vice versa
* **Zero covariance means x and y are not related and are completely independent.**
* **Calculating covariance using Python example:-**

# vector covariance

from numpy import array

from numpy import cov

# define first vector

x = array([1,2,3,4,5,6,7,8,9])

print(x)

# define second covariance

y = array([9,8,7,6,5,4,3,2,1])

print(y)

# calculate covariance

**Sigma = cov(x,y)[0,1]**

**print(Sigma)**

OUTPUT:-

[1 2 3 4 5 6 7 8 9]

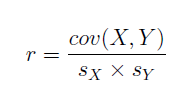
[9 8 7 6 5 4 3 2 1]

-7.5

## **IMPORTANT TERMINOLOGY FOR UNDERSTANDING PCA: CORRELATION :-**

* **The covariance can be normalized to a score between -1 and 1 to make the magnitude interpretable by dividing it by the standard deviation of X and Y . The result is called the**

**correlation of the variables, also called the Pearson correlation coefficient**



**Where r is the correlation coefficient of X and Y , cov(X; Y ) is the sample covariance of X and Y and sX and sY are the standard deviations of X and Y respectively.**

* **NumPy provides the corrcoef() function for calculating the correlation between two variables directly.**
* **Calculating correlation using Python example:-**

vector correlation

from numpy import array

from numpy import corrcoef

# define first vector

x = array([1,2,3,4,5,6,7,8,9])

print(x)

# define second vector

y = array([9,8,7,6,5,4,3,2,1])

print(y)

# calculate correlation

corr = corrcoef(x,y)[0,1]

print(corr)

OUTPUT:-

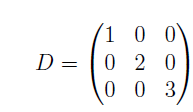
[1 2 3 4 5 6 7 8 9]

[9 8 7 6 5 4 3 2 1]

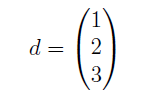
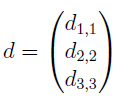
-1.0

## **IMPORTANT TERMINOLOGY FOR UNDERSTANDING PCA: SQUARE MATRIX AND DIAGONAL MATRIX**

* + **A matrix is a square matrix if number of rows (n) = number of columns(m)**
  + **In rectangular matrix number of rows(n) is not equal to number of columns(m)**
  + **A diagonal matrix is one where values outside diagonal have 0 values. Main diagonal is taken from top left of the matrix to bottom right**
  + **A diagonal matrix is denoted by D and may be represented as full matrix or as a vector of values of main diagonal.**
  + **Mentioned below is 3\*3 square diagonal matrix:-**

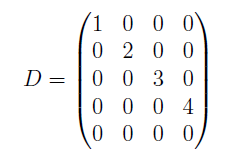


* + **Diagonal matrix denoted as vector**



* + **A diagonal matrix does not have to be square. In the case of a rectangular matrix, the**

**diagonal would cover the dimension with the smallest length; for example:**



* + **NumPy provides the function diag() that can create a diagonal matrix from an existing**

**matrix, or transform a vector into a diagonal matrix as shown below:-**

# diagonal matrix

from numpy import array

from numpy import diag

# define square matrix

M = array([

[1, 2, 3],

[1, 2, 3],

[1, 2, 3]])

print(M)

# extract diagonal vector

d = diag(M)

print(d)

# create diagonal matrix from vector

D = diag(d)

print(D)

OUTPUT:-

[[1 2 3]

[1 2 3]

[1 2 3]]

[1 2 3]

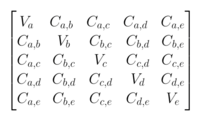
[[1 0 0]

[0 2 0]

[0 0 3]]

## **IMPORTANT TERMINOLOGY FOR UNDERSTANDING PCA: COVARIANCE MATRIX**

* + The **covariance matrix is a square and symmetric matrix that describes the covariance between two or more variables**
  + The **diagonal of covariance matrix are the variances of each random variable and are called as variance-covariance matrix.**
  + A covariance matrix is a generalization of the covariance of two variables and captures the way in which all variables in the dataset may change together.
  + A covariance matrix of dataset having 5 dimensions (columns) -> a,b,c,d,e is denoted as below



**Where Va:- Variance along dimension(column ) a, similarly Vb:- Variance along dimension(column) b, and so on**

**Ca,b : Covariance along dimension a and b**

**Ca,c : Covariance along dimension a and c**

**Ca,d : Covariance along dimension a and d**

**Ca,e : Covariance along dimension a and e**

**And similarly for other covariance value**

* + **If you have a set of n numeric data items, where each data item has d dimensions, then the covariance matrix is a d-by-d symmetric square matrix where there are variance values on the diagonal and covariance values off the diagonal.**

For e.g. if there is a matrix (X) of dimension 5\*3 as shown below

X = array([

[1, 5, 8],

[3, 5, 11],

[2, 4, 9],

[3, 6, 10],

[1, 5, 10]])

**Then covariance of matrix (X) will have dimension 3\*3 as there are 3 columns , here first row will have covariance value of 1st column with itself and 2&3 columns, second row will have covariance value of 2nd column with itself and 1&3 columns and third row will have covariance of 3rd column with itself and first 2 columns.**

* + Covariance of above matrix (X) is shown below which is calculated after transposing the matrix (X)

[[ 1. 0.25 0.75]

[ 0.25 0.5 0.25]

[ 0.75 0.25 1.3 ]]

**Variance of the dimension of matrix (X) i.e. 1st, 2nd and 3rd column are in in main diagonal (1,0.5 and 1.3)**

**Covariance of the dimensions are the off diagonal elements**

* + Formulate to calculate covariance

Cov(*X*, *Y*) = Σ ( *X*i - *X* ) ( *Y*i - *Y* ) / *N* = Σ *x*i*y*i / *N-1*

Where *X*  and *Y*  are the mean value of X and Y datasets respectively

*X*i is the *i*th value in the X dataset

*Y*i is the *i*th value in the Y dataset

*N is number of values in each dataset*

*Example:-*

X Y Z

Height Score Age

64.0 580.0 29.0

66.0 570.0 33.0

68.0 590.0 37.0

69.0 660.0 46.0

73.0 600.0 55.0

mean = 68.0 600.0 40.0

n=5

The covariance matrix for this data set is:

X Y Z

X 11.50 50.00 34.75

Y 50.00 1250.00 205.00

Z 34.75 205.00 110.00

The 11.50 is the variance of X, 1250.0 is the variance of Y, and 110.0 is the variance of Z. For variance, in words, subtract each value from the dimension mean. Square, add them up, and divide by n-1. For example, for X:

Var(X) = [ (64–68.0)^2 + (66–68.0^2 + (68-68.0)^2 + (69-68.0)^2 +(73-68.0)^2 ] / (5-1) = (16.0 + 4.0 + 0.0 + 1.0 + 25.0) / 4 = 46.0 / 4 = 11.50.

The covariance for XY is best shown by example:

Covar(XY) =

[ (64-68.0)\*(580-600.0) + (66-68.0)\*(570-600.0) + (68-68.0)\*(590-600.0) + (69-68.0)\*(660-600.0) + (73-68.0)\*(600-600.0) ] / (5-1) =

[80.0 + 60.0 + 0 + 60.0 + 0] / 4 =

200 / 4 = 50.0

* + **Calculating covariance of a matrix using Numpy**

# covariance matrix

from numpy import array

from numpy import cov

# define matrix of observations

X = array([

[1, 5, 8],

[3, 5, 11],

[2, 4, 9],

[3, 6, 10],

[1, 5, 10]])

print(X)

# calculate covariance matrix

Sigma = cov(X.T)

print(Sigma)

**Output:-**

[[ 1 5 8]

[ 3 5 11]

[ 2 4 9]

[ 3 6 10]

[ 1 5 10]]

[[ 1. 0.25 0.75]

[ 0.25 0.5 0.25]

[ 0.75 0.25 1.3 ]]

## **IMPORTANT TERMINOLOGY FOR UNDERSTANDING PCA: EIGENDECOMPOSITION:-**

* + **Matrix decomposition is used to reduce a matrix to their constituent (component) parts in order to simplify more complex matrix operations.**
  + **Eigendecomposition method is matrix decomposition method that decomposes a square matrix into set of eigenvector and eigenvalues.**
  + **A vector is an eigenvector of a matrix if it satisfies following equation**



**Where:-**

**A= Covariance/square matrix that is getting decomposed**

**v= eigenvector of the covariance/ square matrix**

**= lamda (lower case) is the eigenvalue scalar**

* + **A matrix could have one eigenvector and eigenvalue for each dimension of parent matrix.**
  + **Not all square matrix can be decomposed to eigenvector and eigenvalue and some can only be decomposed in a way that requires complex numbers.**
  + **Parent matrix is product of eigenvector and eigenvalues**



**Where :-**

**Q= matrix composed of eigenvectors.**

**= lambda(upper case) is diagonal matrix composed of eigenvalues**

**= Transpose of matrix that is comprised of eigenvectors.**

* + All vectors change their direction when they are multiplied by A (square matrix), **certain exceptional vectors (x) are in same direction as Ax and these are called as eigen vectors.**

**Multiply an eigenvector(x) by A , the vector Ax is number**  **times the original x . This eigenvalue(****) would specify if vector x is stretched or shrunk or reversed or left unchanged when multiplied by A.**

* + **Eigenvectors are column vector that have length or magnitude equal to 1.0**
  + **Eigenvalues are coefficient applied to eigenvectors that give the vectors their length or magnitude.**
  + **A matrix that has only positive eigenvalues is referred to as positive definitive matrix, where as if the eigenvalues are all negative its referred as negative definitive matrix.**
  + **The eigenvectors are returned as a matrix with the same dimensions as the parent matrix,**

**where each column is an eigenvector, e.g. the first eigenvector is vectors[:, 0] (first column). Eigenvalues are returned as a list, where value indices in the returned array are paired with eigenvectors by column index, e.g. the first eigenvalue at values[0] is paired with the first eigenvector at vectors[:, 0].**

* + **Calculation of EigenDecomposition using numpy using eig() function :-**
* **Import numpy modules:-**

from numpy import array

from numpy.linalg import eig

# define matrix

* **Define square matrix (A):-**

A = array([

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

* **Find eigenvalue and eigenvector**

values,vectors=eig(A)

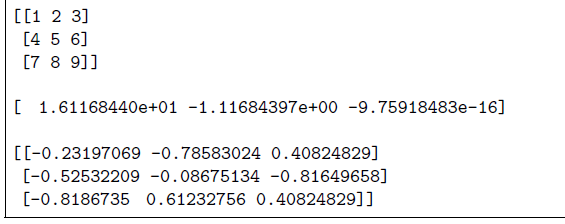
* **Printing square matrix, eigenvalues and vector**

print(A)

print(values)

print(vectors)

**Output:-**



* + **Confirm an Eigenvector and Eigenvalue:-**
* To confirm if an eigen value and eigen vector are correct do the following.
* Fetch eigenvector (vector) and eigenvalue (value) from square matrix(A) . Note eigenvector is a matrix with dimension same as original square matrix (A) and eigenvalue is a list.
* Multiply square matrix (A) with first eigen vector (vector[:,0]) i.e. first column of eigen matrix. Store the resultant vector/ndarray as B
* Multiply first eigen vector (vector[:,0]) i.e. first column of eigen matrix with first eigen value (values[0]) . store the resultant vector/ndarray as C.
* If B and C values same then eigenvector and eigenvalue are same
* Python code

# confirm eigenvector

from numpy import array

from numpy.linalg import eig

# define matrix

A = array([

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

# factorize

values, vectors = eig(A)

**# confirm first eigenvector.Note dot() function in numpy is used to multiply two matrix**

B = A.dot(vectors[:, 0])

print(B)

C = vectors[:, 0] \* values[0]

print(C)

**Output:-**

[ -3.73863537 -8.46653421 -13.19443305]

[ -3.73863537 -8.46653421 -13.19443305]

* + **Reconstruct original square Matrix from EigenValue and EigenVector**
* **Derive eigenvector (Q) and eigenvalue from given square matrix**
* **Inverse the eigenvector(R)**
* **Arrange the eigenvalue as diagonal matrix (L)**
* **Now multiply Q (eigebvector/matrix), R (inverse of eigen matrix), and L(diagonal matrix of eigenvalue)**
* **Python Code:-**

# reconstruct matrix

from numpy import diag

from numpy.linalg import inv

from numpy import array

from numpy.linalg import eig

# define matrix

A = array([

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

print(A)

# factorize

values, vectors = eig(A)

# create matrix from eigenvectors

Q = vectors

# create inverse of eigenvectors matrix

R = inv(Q)

# create diagonal matrix from eigenvalues

L = diag(values)

# reconstruct the original matrix

B = Q.dot(L).dot(R)

print(B)

**Output:-**

[[1 2 3]

[4 5 6]

[7 8 9]]

[[ 1. 2. 3.]

[ 4. 5. 6.]

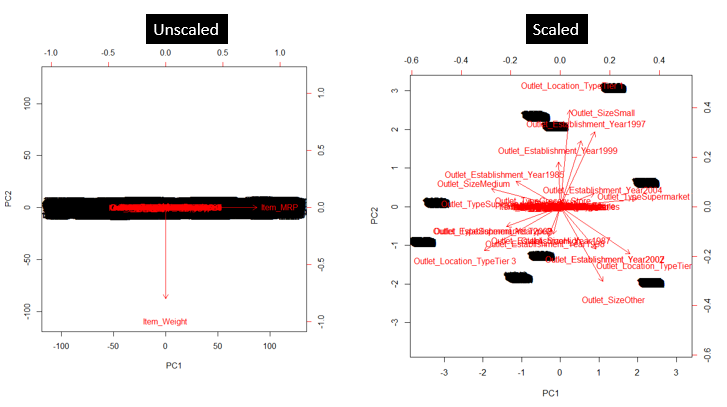
[ 7. 8. 9.]]

## **PCA INTRODUCTION:-**

* + **What happens when a data set has too many variables? Here are few possible situations which you might come across:**
    - * **Modeling data with more features (columns/variables/dimensions) is challenging and if model is built with more irrelevant features are often less skillful than model built with relevant data.**
      * **Start thinking of some strategic method to find few important variables to increase the accuracy of the model**
  + **In real world we use multi-dimension data and as the dimension of the data increases, it increases the computation so it is required to reduce the dimension by removing redundant dimensions or keeping just the important dimension**
  + Principal component analysis (PCA) is a **statistical method for data reduction techniques used to reduce number of variables or dimension in a dataset ( also called as data reduction techniques). Other way of saying is PCA is a statistical method of extracting important variables (in form of component) from large set of variables available in a dataset.**
  + **During data analysis as we try to find patterns among the data so we want data to be spread out across each dimension and want the dimensions to be independent such that if the data has high covariance when represented in some n number of dimension then we replace those dimensions with linear combination of those n dimensions.** **Now that data will only be dependent on linear combination of those related (high covariance) n dimensions.**

## **PRE-REQUISISTE FOR PCA:-**

* + **PCA is always done on scaled or standardized or normalized dataset.The principal components are supplied with normalized version of original predictors (variables /input dataset). 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. Performing PCA on un-normalized variables will lead to insanely large loadings for variables with high variance. In turn, this will lead to dependence of a principal component on the variable with high variance. This is undesirable as there will be misleading components.**
  + As shown in image below, PCA was run on a data set twice (with unscaled and scaled predictors). This data set has ~40 variables. You can see, first principal component is dominated by a variable Item\_MRP. and, second principal component is dominated by a variable Item\_Weight. This domination prevails due to high value of variance associated with a variable. When the variables are scaled, we get a much better representation of variables in 2D space.



* + **PCA is always done on symmetric correlation or covariance matrix. This means matrix or dataset for PCA should always be numeric.**
  + **Since PCA can be applied only on numerical data, hence if there is any categorical data in dataset it must be converted to numerical.**
  + **Data cleaning and imputation needs to be done before implementing PCA.**

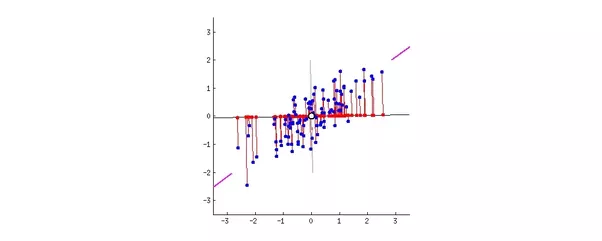
## **WHAT PCA DOES AND ILLUSTRATION:-**

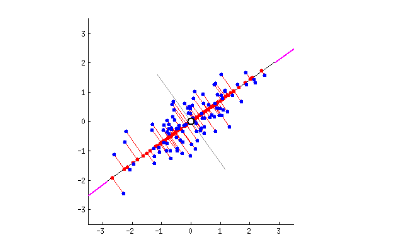
* + **In simple words, principal component analysis is a method of extracting important variables (in form of components) from a large set of variables available in a data set. It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible. With fewer variables, visualization also becomes much more meaningful. PCA is more useful when dealing with 3 or higher dimensional data .**
  + The core of PCA is **matrix factorization method from linear algebra.**
  + **Find linearly independent dimensions which can losslessly represent the data points.**
  + PCA is done by **combining/uniting highly correlated variables together** , **but this is done at the cost of accuracy. We can define principal component as linear combination of original predictors in a dataset. In the image below PC1 and PC2 are principal components.**

PCA finds new sets of dimensions (or set of basic of views) such that all the dimensions are orthogonal (and hence are linearly independent) and dimensions are ranked according to the variance of the data (spread of the data) long them (refer to PC1 and PC2 in image below which is orthogonal to each other). This means more important principle axis/component occurs first (more important =more variance/spread of data). Prediction is done on this new set of dimension with minimum projection error (sum of perpendicular distance of the actual data points from the newly projected dimension line /principal axis is total projection error)

**Notes on Projection error (IMP):-**

Suppose we have to transform a 2 dimensional representation of data points to a one dimensional representation. So we will basically try to find a straight line and project data points on them. (A straight line is one dimensional). There are many possibilities to select the straight line. Lets see two such possibilities –



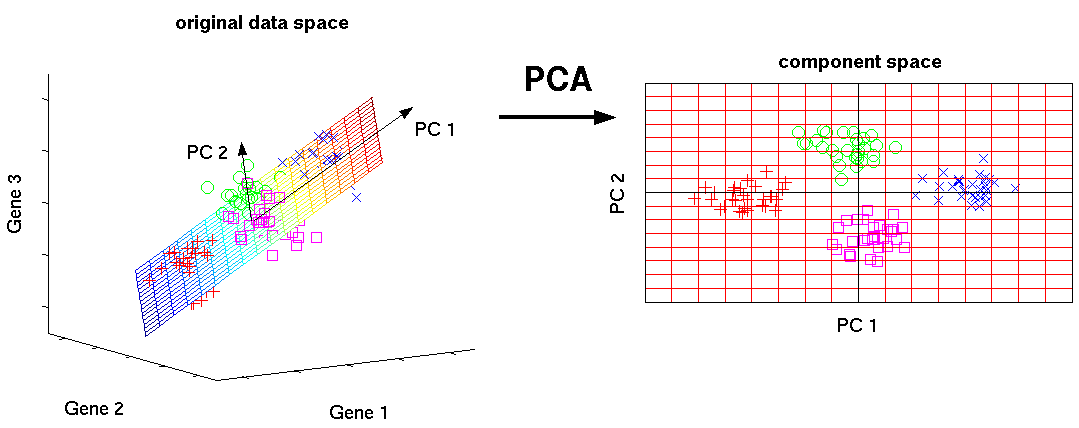


Say magenta line will be our new dimension.If you see the red lines (connecting the projection of blue points on magenta line) i.e. the perpendicular distance of each data point from the straight line is the projection error. Sum of the error of all data points will be the total projection error.  
Our new data points will be the projections (red points) of those original blue data points. As we can see we have transformed 2 dimensional data points to one dimensional data points by projection them on 1 dimensional space i.e. a straight line. That magenta straight line is called *principal axis.*Since we are projecting to a single dimension, we have only one principal axis

Clearly, Second choice of straight line is better because -  
\* The projection error is less than that in the first case.  
\* Newly projected red points are more widely spread out than the first case. i.e. more variance.

**The above mentioned two points are related i.e. if we minimize the reconstruction error, the variance will increase.**

* + **ILLUSTRATION:-**
* 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 and hence is a tedious job to do EDA.**
* **In this case, it would be a lucid approach to select a subset of *p* *(p << 50)* predictor which captures as much information. Followed by plotting the observation in the resultant low dimensional space**
* **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**



* **The principal component can be written as:**

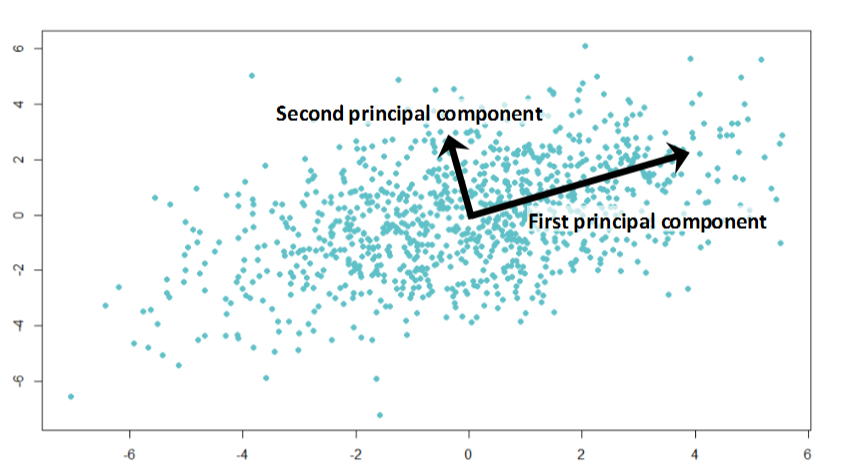
Z¹ = Φ¹¹X¹ + Φ²¹X² + Φ³¹X³ + .... +Φp¹Xp

**where,**

* + - Z¹ is first principal component
    - Φp¹ is the loading vector comprising of loadings (Φ¹, Φ²..) of first principal component. **The loadings are constrained to a sum of square equals to 1. This is because large magnitude of loadings may lead to large variance. It also defines the direction of the principal component (Z¹) along which data varies the most**. It results in a line in p dimensional space which is closest to the n observations. Closeness is measured using average squared euclidean distance.
    - X¹..Xp are normalized predictors. **Normalized predictors have mean equals to zero and standard deviation equals to one.**
  + **First principal component** is a linear combination of original predictor variables which captures the maximum variance in the data set. It determines the direction of highest variability in the data. Larger the variability captured in first component, larger the information captured by component. No other component can have variability higher than first principal component.The first principal component results in a line which is closest to the data i.e. it minimizes the sum of squared distance between a data point and the line.
  + **Second principal component** (Z²) is also a linear combination of original predictors which captures the remaining variance in the data set and is uncorrelated with Z¹. In other words, the correlation between first and second component should is zero. It can be represented as:

Z² = Φ¹²X¹ + Φ²²X² + Φ³²X³ + .... + Φp2Xp

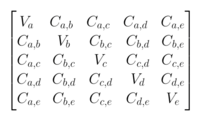
* + If the two components are uncorrelated, their directions should be orthogonal (image below). This image is based on a simulated data with 2 predictors. Notice the direction of the components, as expected they are orthogonal. This suggests the correlation b/w these components in zero.



* + succeeding principal component follows a similar concept i.e. they capture the remaining variation without being correlated with the previous component. In general, for *n × p* dimensional data, min(*n-1, p)* principal component can be constructed.
  + **The directions of these components are identified in an unsupervised way i.e. the response variable(Y) is not used to determine the component direction. Therefore, it is an unsupervised approach.**
* **PCA is an analysis approach. One can do PCA using SVD(singular value decomposition) , or can do PCA doing the eigen-decomposition** , or many other methods. SVD is just another numerical method. So, don’t confuse the terms PCA and SVD.

## **PCA IMPLEMENTATION STEP BY STEP PROCESS (USING EIGEN DECOMPOSITION METHOD):-**

* + - * **Calculate the covariance matrix X of data points. Covariance matrix is derived because the covariance matrix is symmetric or square and Eigendecomposition works on symmetric matrix.**



**If we refer to above covariance matrix, main diagonal element contains variance of dimension and off diagonal elements contains the covariance value between dimensions. The objective is to have high variance data(more spread) and remove correlated dimensions i.e. covariance value among the dimension should be zero (linearly independent) therefore for PCA we should have covariance matrix that have large numbers in diagonal elements and zero values in off diagonal elements and this matrix is called as diagonal matrix. The process of transforming a matrix to diagonal matrix is called diagonalization.**

* + - * **Calculate the eigen vectors and corresponding eigen values**
      * **Sort the eigen vectors according to thier eigen values in decreasing order.**
      * **Choose first k eigen vectors and that will be new k dimensions.**
      * **Transform the original n-dimensional data points into k dimensions.**

## **STEP BY STEP IMPLEMENTON OF PCA USING PYTHON :-**

**BELOW ARE THE STEPS FOR IMPLEMENTING PCA STARTING FROM READING THE RAW DATA UNTIL CREATING A MODEL**

* + - * **Remove dependent and identifier variables from the dataset. Note identifier variables are categorical variables that have single individual per category and are used as unique identifiers (e.g. ID number, passport no, Social security No)**
      * **Do data cleaning like imputation of missing values**
      * **Convert categorical field (if any) to numerical field (dumify)**
      * **Check if there is any missing value using below syntax:-**

***input\_dataframe.isnull().sum()***

***All the column should have value=0***

***Where input\_dataframe is the dataframe created until step-3 after cleaning and removing unwanted variables***

* + - * **Scale or standardize or normalize the input data before PCA to remove different variance across all variables using below syntax**

***from sklearn.preprocessing import StandardScaler***

***scale=StandardScaler()***

***input\_dataframe= scale.fit\_transform(input\_dataframe)***

* + - * **Check the standard deviation and mean of the *input\_dataframe after standardizing the input dataset***

**print(*input\_dataframe*.std())**

**print(*input\_dataframe*.mean())**

**Normalized predictors have mean equals to zero and standard deviation equals to one.**

* + - * **Creating Covariance Matrix of *input\_dataframe* which is now scaled is as shown below**

***cov\_mat = np.cov(input\_dataframe.T)***

***print(' covariance matrix for input dataset for PCA: \n%s' %* cov\_mat*)***

* + - * **Perform Eigen decompsition on covariance matrix i.e. derive eigen values and eigen vectors**

**eig\_vals, eig\_vecs = np.linalg.eig(cov\_mat)**

**print('Eigenvectors \n%s' %eig\_vecs)**

**print('\nEigenvalues \n%s' %eig\_vals)**

* + - * **Print the Eigen values in descending order .**

***eig\_pairs = [(np.abs(eig\_vals[i]), eig\_vecs[:,i]) for i in range(len(eig\_vals))]***

***print('Eigenvalues in descending order:')***

***for i in eig\_pairs:***

***print(i[0])***

**#NOTE-1:- Number of eigen values printed will be equal to number of columns in the dataset**

**#NOTE-2:- The eigenvectors with the lowest eigenvalues describe the least amount of variation within the dataset. Therefore, these values can be dropped**

**#NOTE-3:- In above code if FIRST 14 PRINCIPAL COMPONENTS REPRESENT AROUND 99% VARIATION OF DATA. SO WE CAN CONSIDER FIRST 14 PRINCIPAL COMPONENTS FOR MODEL INPUT**

* + - * **Derive Explained variance ratio of *input\_dataframe* which is now scaled . Plot (scree) based on number of principal components vs cumulative explained variance. A scree plot displays the variance explained by each principal component within the analysis**

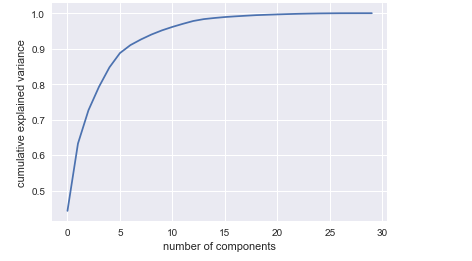
***pca=PCA().fit(input\_dataframe)***

***plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))***

***plt.xlabel('Number of components')***

***plt.ylabel('Cumulative explained variance')***

***plt.show()***



**NOTE-1:- In above plot after 14 PRINCIPAL COMPONENTS THE VARIATION HAS DECREASED OR THE DATA IS CONSTANT. THE PLOT JUSTIFIES THE NUMBER OF COMPONENT SELECTED IN POINT-9 (REFER TO NOTE-3)**

**# PRINTING ALL explained\_variance\_ratio\_ AND CUMULATIVE explained\_variance\_ratio\_FOR THE DATASET**

**print (pca.explained\_variance\_ratio\_)**

**print (np.cumsum(pca.explained\_variance\_ratio\_))**

* + - * **TRANSFORM THE input\_dataframeWHICH IS SCALED DATA TO A NEW DATASET (model\_ input\_dataframe*) WHICH WILL BE USED AS INPUT TO DIFFERENT MODELS*  BY CONSIDERING ONLY FIRST ‘n’ PRINCIPAL COMPONENTS OF SCALED INPUT DATASET (input\_dataframe) AS THESE ‘n’ PRINCIPAL COMPONENTS HAS THE MAX VARIATION (>90%) OF INPUT DATASET**

***model\_ input\_dataframe* =PCA(n\_components=14).fit\_transform(*input\_dataframe*)**

***model\_ input\_dataframe***

**NOTE-1:-Here we have considered n\_components=14 because first 14 components (based on decreasing order) has 99% variation of data (refer to point-9)**

**NOTE-2:- We can check the explained variance ratio of 14 principal components using command as shown below:-**

***pca = PCA(n\_components=14)***

***print(pca.explained\_variance\_ratio\_)***

* + - * **Divide the PCA dataset created in step-11 (model\_ input\_dataframe) into test and train for modeling as shown below:-**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split(*model\_ input\_dataframe*,y,test\_size=0.20,random\_state=123)**

**print(x\_train.shape)**

**print(y\_train.shape)**

**print(x\_test.shape)**

**print(y\_test.shape)**

**here y is the target**

* + - * **<WRITE NEXT STEPS ON MODELING>**
      * **Note:- Step7,8,9 can be ignored but is good to have in steps**

## **REFERENCE FOR PCA:-**

<https://www.youtube.com/watch?v=BfTMmoDFXyE>

<https://www.youtube.com/watch?v=_UVHneBUBW0>

<https://www.youtube.com/watch?v=N9MRzIHyA_Q>

<https://medium.com/@aptrishu/understanding-principle-component-analysis-e32be0253ef0>

<https://medium.com/district-data-labs/principal-component-analysis-with-python-4962cd026465>

<https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/>

<https://jamesmccaffrey.wordpress.com/2017/11/03/example-of-calculating-a-covariance-matrix/>