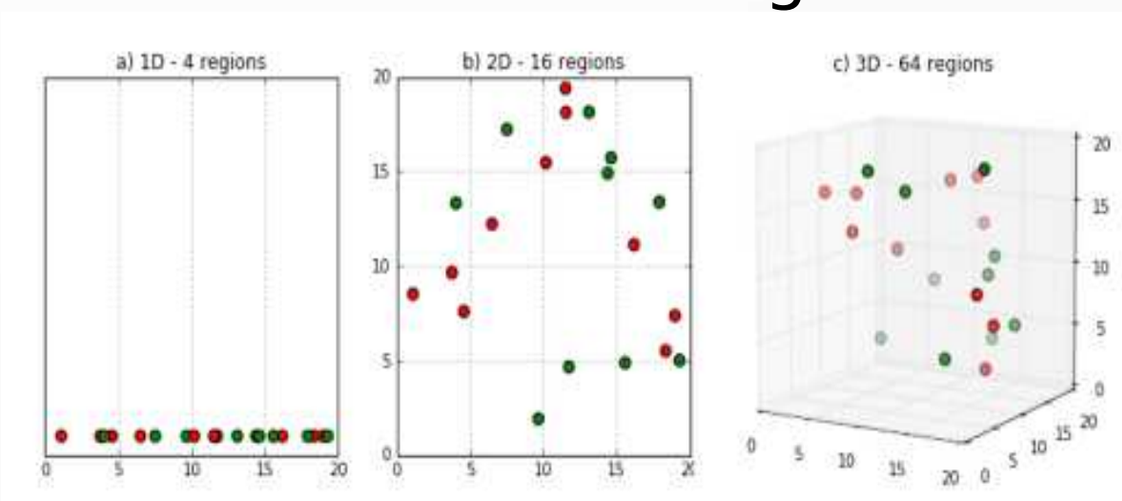
4. Reduce Dimensionality

Typical Dimensionality in Data

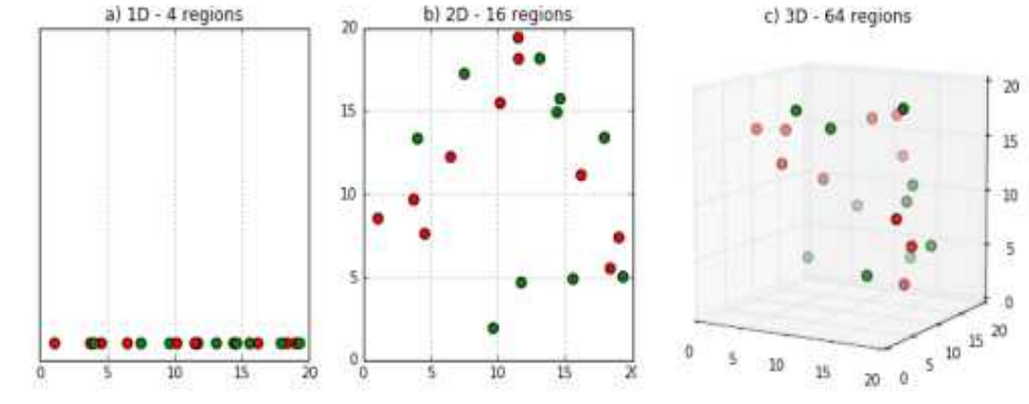
- There are many issues that arise when analysing and organising data in high-dimensional spaces
- Typical dimensions of data:
  - **[Text data]**
  - if you crawl a news website to have one week's news, DIM>10000
  - it is dictionary size you have built based on the words (We need to represent each document based on the these words in a dictionary)
  - **[Image data]**: 64×64 image would have 4096 dimensions!
  - **[Genomic data]**: Parkinson case-control data has 408,803 Single-nucleotide polymorphisms (SNPs), & Alzheimer has 380,157 SNPs.

Curse of Dimensionality

- In machine learning we face unique problems when analysing and organising data in high-dimensional spaces
- When the dimensionality increases, the volume of the space increases so fast that the available data become sparse.
- This is really problematic since there is not enough data locally
- 1D, 2D, 3D space data



Curse of Dimensionality...



- In the figure, first we had observed some points in 1D data in 4 regions (20 divided by 5).
- Subsequently these points are transferred into 2D space, into a 16 region space.
- In the next step the points are in a 3D space with 64 regions.

- What would happen when we get to 100 dimensions?
  - At its core, the curse of dimensionality, dictates that as the number of dimensions increases, the number of regions grows exponentially
  - That makes our data sparse and somehow not useful anymore
  - Also some of our intuitions from low dimensional spaces fail badly in high dimensions

Curse of Dimensionality...

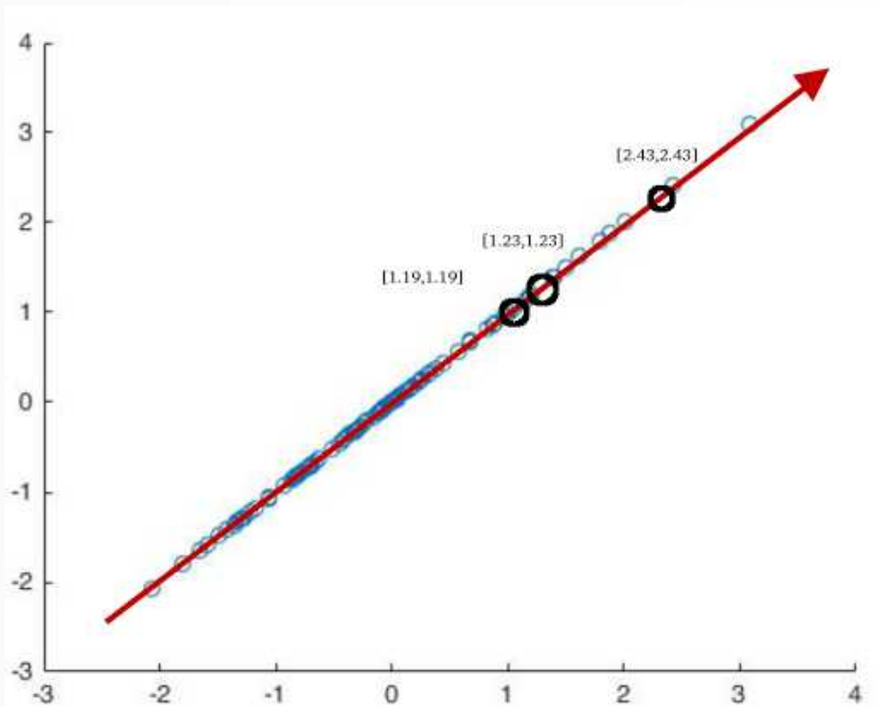
- Concentration effect
  - Relative contrast between near and far neighbours diminishes as the dimensionality increases.
- This problem can imply that
  - Clustering or KNN algorithms may be meaningless in high dimensions.
  - However, there might still be patterns in high dimension. We just need better distance metrics. So Research is on!
  - Until we develop better distance metrics, we should aim to reduce the dimensionality where possible

Solving Curse of Dimensionality

- In some problems, there are too many variables.
  - Are all variables important?
    - If not then
      - some of them are irrelevant
      - can be removed
- If all variables are numeric, what if they are correlated?
  - This means redundancy!
  - Can we club them together?

Solving Curse of Dimensionality...

- Dimensionality reduction refers to the process of converting a dataset of dimension into dimension where ensuring similar information contents.



- We can take a subset of data from this graph, which looks like this

$$X = \begin{bmatrix} 1.19 & 1.19 \\ 1.23 & 1.23 \\ 2.43 & 2.43 \end{bmatrix}$$

- is the first and second features of these data points are the same?
  - why not just to use only one of these features?

Dimensionality Reduction

Solving Curse of Dimensionality...

$$X = \begin{bmatrix} 1.19 & 1.19 \\ 1.23 & 1.23 \\ 2.43 & 2.43 \end{bmatrix}$$

So, we can transform this data points as the only dimension by using a projection vector:

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- By multiplying data points and projection vector we will have:

$$\begin{bmatrix} 1.19, 1.19 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 1.19$$

$$\begin{bmatrix} 1.23, 1.23 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 1.23$$

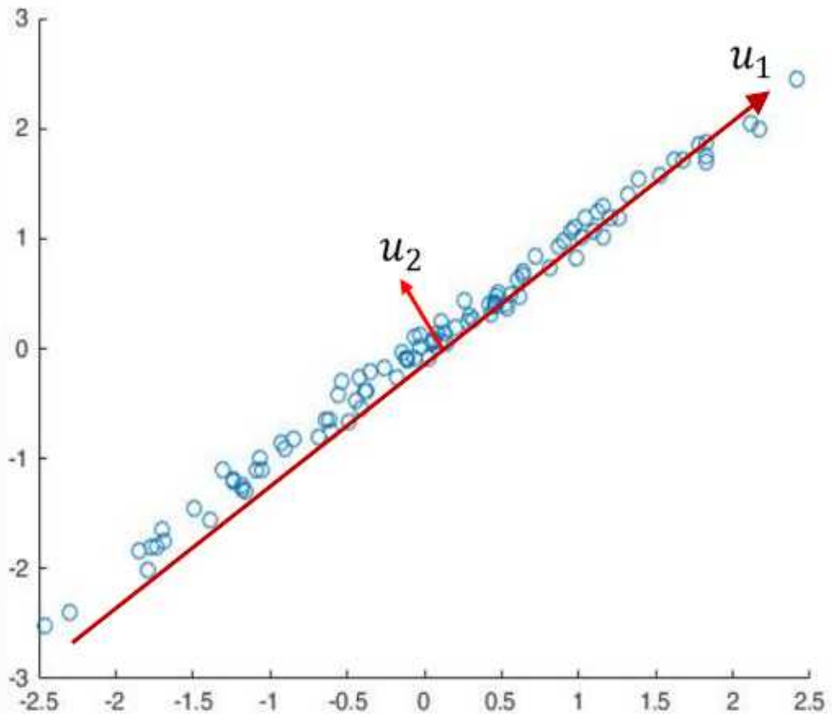
$$\begin{bmatrix} 2.43, 2.43 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 2.43$$

$$\Rightarrow X' = \begin{bmatrix} 1.19 \\ 1.23 \\ 2.43 \end{bmatrix}$$

- X’ is the projected data into a single dimension (the red arrow in the figure).
- We have just reduced one dimension of this 2D data.
- If you notice the formation of the data, you can see that is also the direction of maximum variance in data!

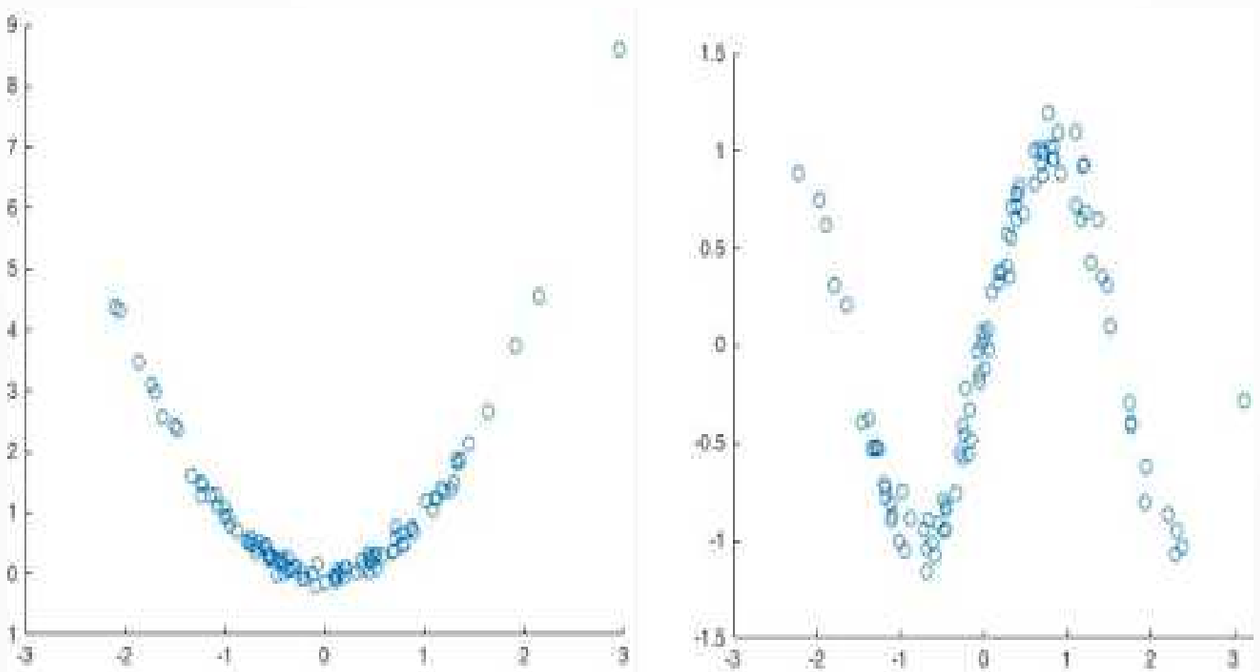
Solving Curse of Dimensionality...

- What if the data is not exactly on the red arrow? Consider this example.
- As you can see in the figure:
  - The dimension vector, points towards the direction of the highest variance
  - The dimension vector, points towards the lowest variance in the subspace, orthogonal to the vector
- Thus, projecting onto maximum variance direction () means capturing more variance and results in capturing more information to analyse.



Solving Curse of Dimensionality...

- There are also some examples in which the points lie on noisy curves and shapes



- In this unit, we will confine ourselves to linear dimensionality reduction problems only.

Principal Component Analysis (PCA)

Preliminaries
Formulation
Implementation

Preliminaries

- The goal of PCA is to take data points in dimensions, which may be correlated, and summarises them by a new set of uncorrelated axes.
  - The uncorrelated axes are called principal components or principal axes.
  - These axes are linear combinations of the original dimensions.
  - The first components capture as much of the variation (or variance) among the data points as possible.

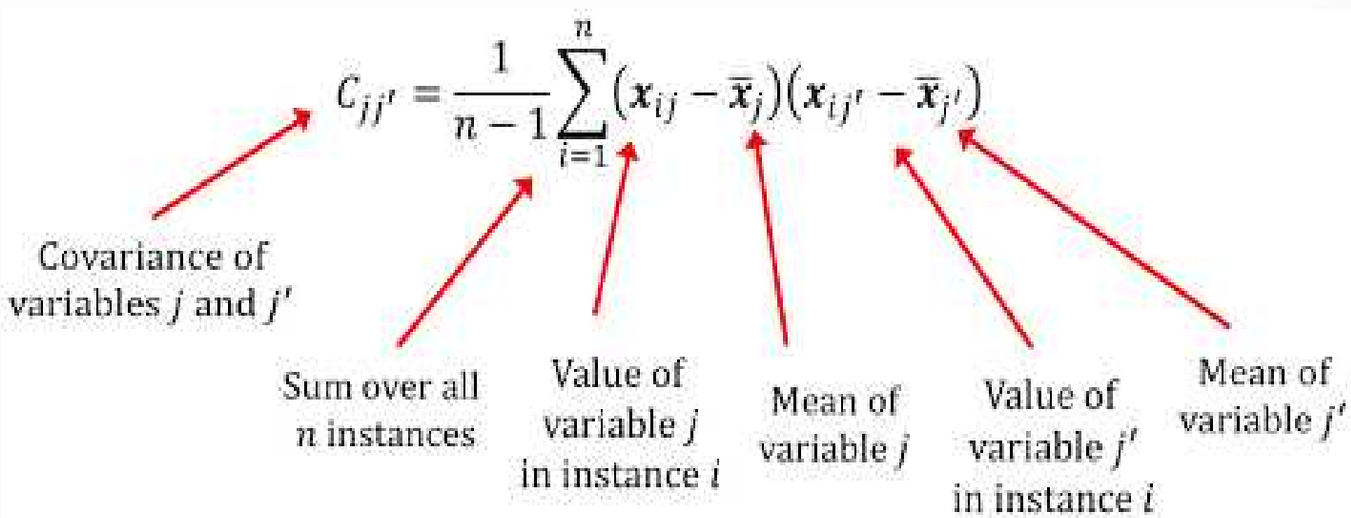
Preliminaries...

- Variance across each variable
  - Data is represented as a cloud of points in a multidimensional space with one axis for each of the variables
  - The centroid of the points is defined by the mean of each variable
  - The variance of each variable j is the average squared deviation of its n values around the mean of that variable

$$C_{jj} = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2$$

Preliminaries...

- Covariances among variables
  - covariance is a measure of how changes in one variable are associated with changes in a second variable.
  - Degree to which the variables are linearly correlated is represented by their co-variances:



Preliminaries...

- Covariance Matrix
  - The covariance matrix is a matrix that contains variances of all variables on the diagonal and co-variances among all pairs of variables in the off-diagonal entries.
  - It can be written as: $$C = \left(\frac{1}{n-1}\right)\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T$$
  - Let us say

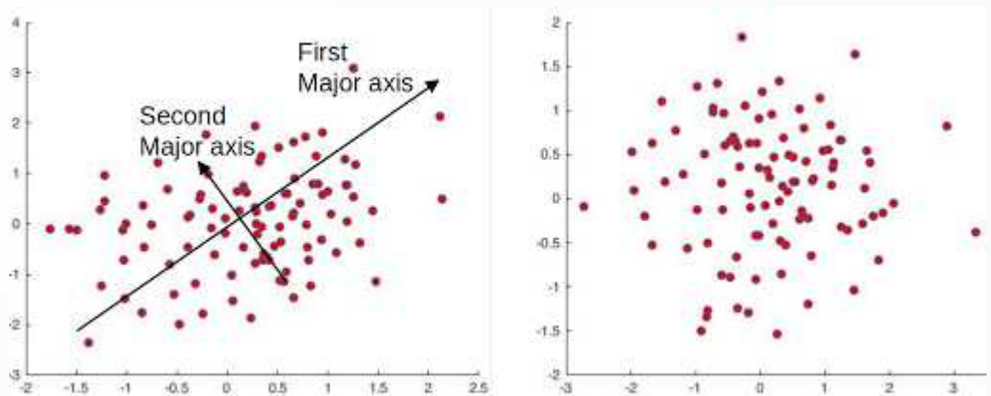
$$y_i = x_i - \bar{x},\ and\ \begin{bmatrix} y_1^T \\ \vdots \\ y_n^T \end{bmatrix},\ then\ C = \left(\frac{1}{n-1}Y^TY\right)$$

Preliminaries...

- PCA: decorrelation
  - The main objective of PCA is to rigidly rotate the axes of t−dimensional axes to a new set of axes (called principal axes) that have the following properties:
    - Ordered such that principal axis- captures the highest variance, axis-2 captures the next highest variance, …. , and axis−d has the lowest variance
    - Covariance among each pair of the principal axes is zero (the principal axes are uncorrelated i.e. they are orthogonal to each other). This is called *decorrelation property*.

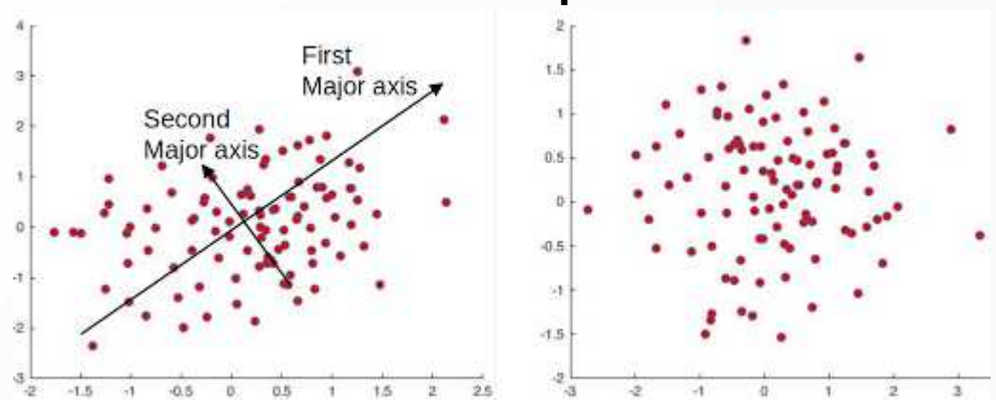
Implementation of PCA

- Consider the following figure as a real-world example
  - First major axis is the direction of largest variance direction
  - The second major axis is the direction of second largest variance.
  - If we calculate the values of these two axis and call them and , we can find the projected data by multiplying and (right panel of the Figure)
    - The projected data lost its correlated form and looks uncorrelated
  - Remember in this example we did not perform any dimensionality reduction



Implementation of PCA...

- Consider the following figure as a real-world example
  - Both projected data and the original data had 2 dimensions
  - So in this particular example, we used PCA to de-correlate the dimensions



  - the covariance matrix of the original is $C = \begin{bmatrix} 0.6592 & 0.2538 \\ 0.2538 & 0.9864 \end{bmatrix}$
  - Which as you can see captures some relations and correlations among the dimensions.
  - On the other hand, the covariance matrix of the projected data is $C = \begin{bmatrix} 1.1247 & 0 \\ 0 & 0.5208 \end{bmatrix}$

Implementation of PCA...

- Which illustrates two important points:
  - There are no correlations among the projected data $C(i,j) = 0; \quad i \neq j$
  - The first dimension or feature has a higher value which means it is more important than the second one $C(1,1) > C(2,2)$
  - Now if we decide to drop one of the dimensions and use dimensionality reduction by PCA, we should choose the eigenvector corresponding to the eigenvalue =1.1247 and then project all data on that axis.
  - Mean square error based on this approximation would be the sum of the remaining eigenvalues.

Python Programming

Setup
PCA using in-built functions

Python program
Setup

- The main objective of PCA is orthogonal transformation of given data into its principal components.
- Here, principle component = axis of maximum variation.

```
# import the necessary modules here
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from sklearn import metrics
```

PCA using Inbuilt Functions in Python

- We start by reading in a data file that contains 5 dimensions or features, download this CSV, add it to your data store and rename it. As with the previous example, we normalise the data, perform PCA and measure the reconstruction error in the recovered data.

```
# Read the data

data = pd.read_csv('data/dataset5DGaussian.csv', delimiter=',', header=None).val
print(data.shape)
```

```
(200, 5)
```

PCA using Inbuilt Functions in Python

- Step 1: Data Normalization
  - Our data consists of 200 datapoints and 5 dimensions (features). Next step is to normalise the data. We use the inbuilt function called scale() function from sklearn.preprocessing to do this.

```
#normalize our data
from sklearn.preprocessing import scale
Xnorm = scale(data)
```

PCA using Inbuilt Functions in Python

- Step 2: Implement PCA
  - We use the inbuilt function to perform PCA. Here, n_components specify the number of principal components to use. We start by using all the principal components.

```
#perform PCA using sklearn PCA implementation

from sklearn.decomposition import PCA
pca = PCA(n_components=5)
pca.fit(Xnorm)
```

```
PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

PCA using Inbuilt Functions in Python

- Step 2: Implement PCA
  - Dimensionality Reduction: How to choose the dimensions to keep
  - To put it differently, how can we choose the number of principal components to retain? We can decide this by looking the variance captured by each principle component.

```
#The amount of variance that each PC explains
var= pca.explained_variance_ratio_
print(var)
```

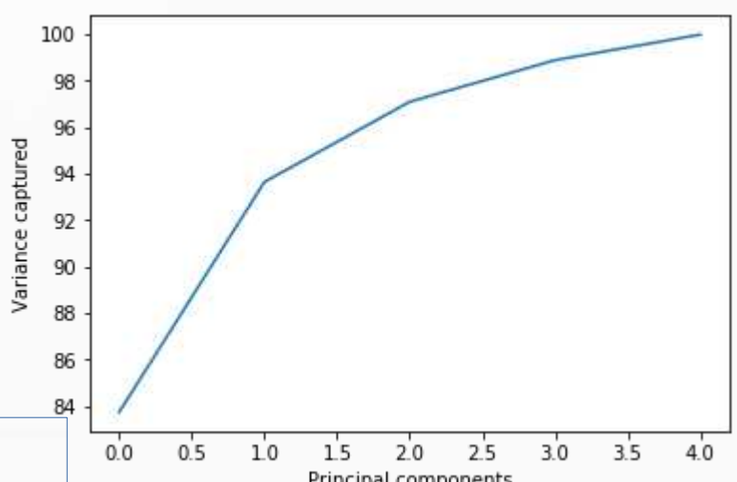
```
[0.83743065 0.09909097 0.0345153  0.01793089 0.01103218]
```

PCA using Inbuilt Functions in Python

- Step 2: Implement PCA
  - Here, we see that the first component captures around 84% variance, second component captures around 10% variance and so on.
  - To make it much easier, we can calculate the cumulative variance:

```
#Cumulative Variance explains
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
print(var1)
plt.plot(var1)
plt.xlabel("Principal components")
plt.ylabel("Variance captured")
```

```
[83.74 93.65 97.1  98.89 99.99]
```



PCA using Inbuilt Functions in Python

- Step 2: Implement PCA
  - So, if k is the number of principal components, we see that k=1 captures around 84% variance, k=2 (the first 2 components together) capture around 94% variance and so on.
  - Since k=2 captures more than 90% variance in our data, lets drop the third, fourth and fifth components.

```
pca = PCA(n_components=2)
Zred = pca.fit_transform(Xnorm)
print(Zred.shape)
```

```
(200, 2)
```

PCA using Inbuilt Functions in Python

- Step 3: Measuring ‘reconstruction error’
  - We can recreate our original data (Xrec) from the reduced data (Zred) using the inverse_transform() function, and calculate the reconstruction error as before.

```
# Reconstruct our data
Xrec = pca.inverse_transform(Zred)
print(Xrec.shape)
```

```
(200, 5)
```

```
# Measure the reconstruction error
rec_error = np.linalg.norm(Xnorm-Xrec, 'fro')/np.linalg.norm(Xnorm, 'fro')
print(rec_error)
```

```
0.2519491448490853
```

PCA using Inbuilt Functions in Python

- Step 3: Measuring ‘reconstruction error’
  - Influence of Dimensionality Reduction on Reconstruction error

```
nSamples, nDims = Xnorm.shape

# vary principal components from 1 to 5
n_comp = range(1,nDims+1)
print(n_comp)
```

```
[1, 2, 3, 4, 5]
```

PCA using Inbuilt Functions in Python

- Step 3: Measuring ‘reconstruction error’
  - Influence of Dimensionality Reduction on Reconstruction error

```
# Initialize vector of rec_error
rec_error = np.zeros(len(n_comp)+1)

for k in n_comp:
    pca = PCA(n_components=k)
    Zred = pca.fit_transform(Xnorm)
    Xrec = pca.inverse_transform(Zred)
    rec_error[k] = np.linalg.norm(Xnorm-Xrec, 'fro')/np.linalg.norm(Xnorm, 'fro')
    print("k={}, rec_error={}".format(k, rec_error[k]))

rec_error = rec_error[1:] #we started recording from index 1, so drop index 0


#Visualize the change in error
plt.plot(n_comp,rec_error)
plt.xlabel('No of principal components (k)')
plt.ylabel('Reconstruction Error')
```

```
k=1, rec_error=0.40319889124
k=2, rec_error=0.251949144849
k=3, rec_error=0.170185410218
k=4, rec_error=0.105034200931
k=5, rec_error=5.94441912765e-16
```