PCA

Principle component Analysis] Feature Extraction

↓

Reduce curse of Dimensionality

- Dimensionality Reduction.

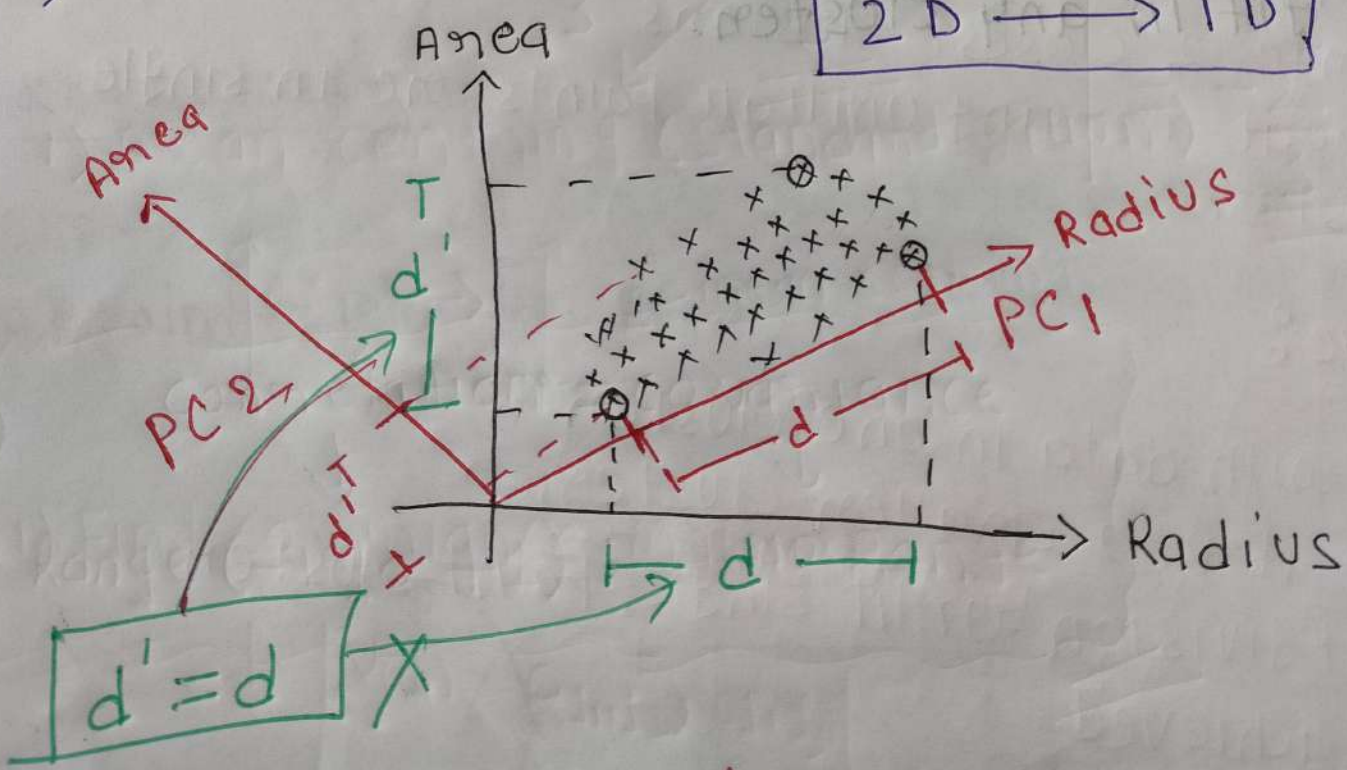
PCA is used in dimensionality Reduction. It's used to simplify & transform high Dimensional data into lower dim. space while retaining as much as original variance as possible.

↳ It identifies & extract most important features, called Principle components.

↳ Dimensionality Reduction
↳ easy Data visualization
↳ Noise Reduction.

Transform

2D ⟶ 1D



d' < d ✓

PC1 > PC2

PCA working :

- PCA work by finding a set of orthogonal (unconnected) axes, known as principle component.
- First PC captures most variance in data. & subsequent components capture decreasing Amt. of variance.
- These components are linear combinations of original features.

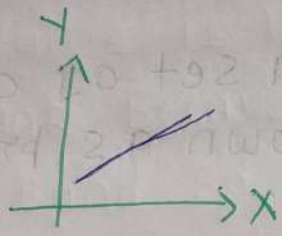
Python Implementation :

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(data)
New_data = pca.transform(data)
```

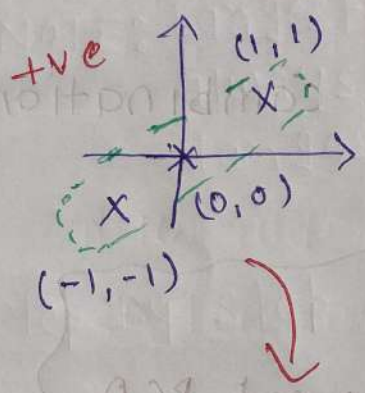
PCA Based on some mathematical concepts :

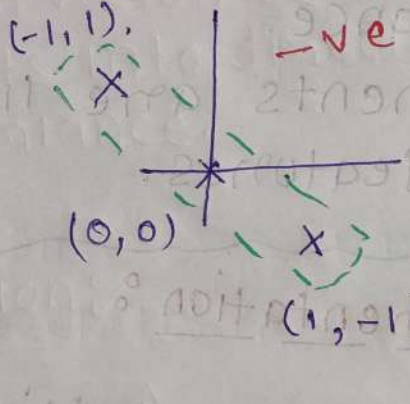
- variance & covariance
- Eigenvalues & Eigenvectors.

covariance →



It determine Relationship b/w ~~of~~ X & Y.





$$\text{Covariance} = \frac{(-1*-1)+(0*0)+(1*1)}{3} = \frac{1+0+1}{3} = \frac{2}{3}$$

$$\frac{(1*-1)+0+(-1*1)}{3} = \frac{-1+0-1}{3} = \frac{-2}{3}$$

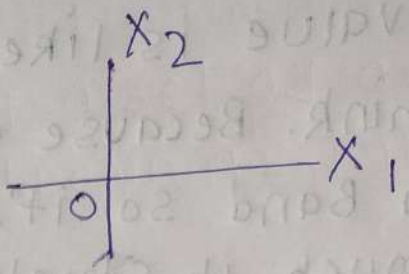
Correlation = covariance

↓ Range (-1 to +1)

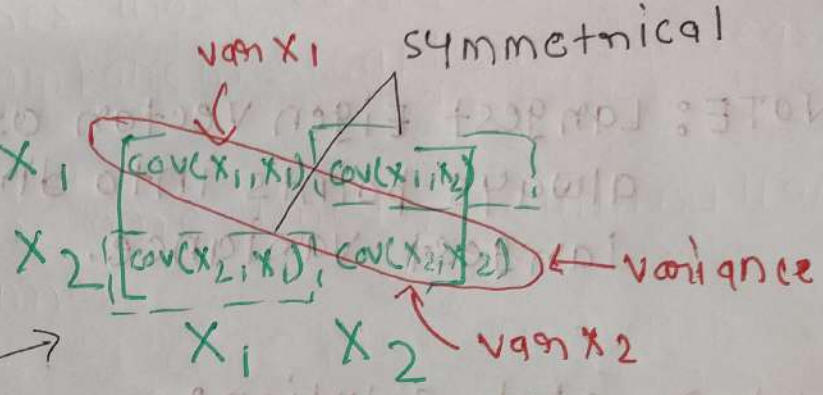
↓ No Range

Covariance matrix :

2D Data = | X_1 | X_2 |



matrix = (2X2)



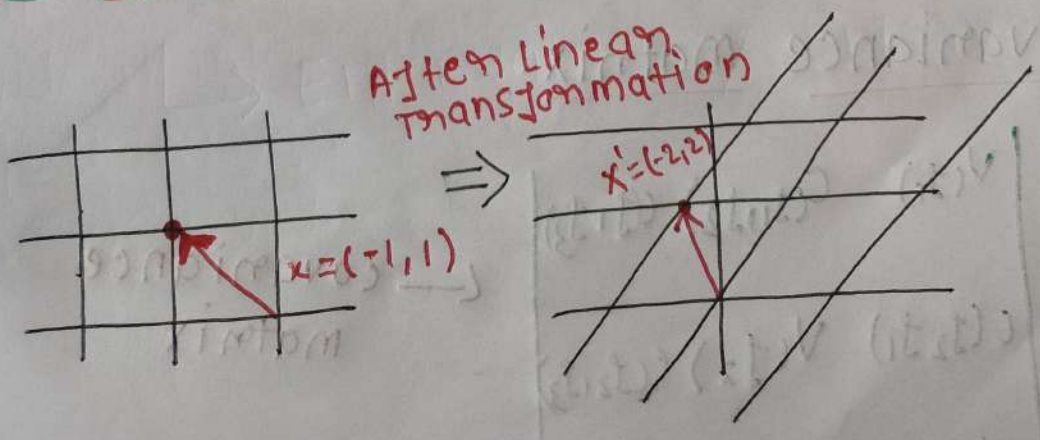
Spread & orientation

variance → spread

covariance → orientation

Eigen vectors :

[change in magnitude But same Direction, No change in direction.]

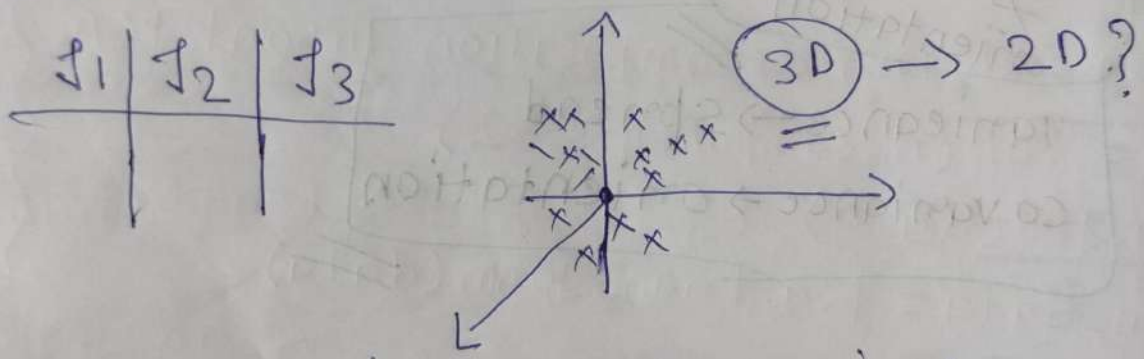


Eigen value:

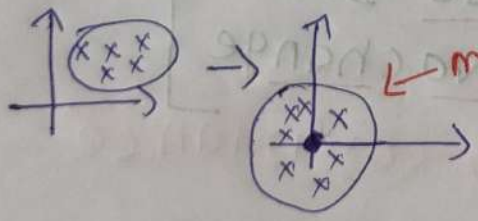
Eigen value is like Rubber Band stretches or shrink. Because whenwe stretch a Rubber Band so it's eigen value tell you How much it stretched.

NOTE: Largest Eigen vecton of cov. matrix always points into direction of largest variance.

Step By step solution:



i) mean centring (standardization)



ii) Find covariance matrix.

$$\begin{array}{c|ccc} & f_1 & f_2 & f_3 \\ f_1 & V(f_1) & C(f_1,f_2) & C(f_1,f_3) \\ f_2 & C(f_2,f_1) & V(f_2) & C(f_2,f_3) \\ f_3 & C(f_3,f_1) & C(f_3,f_2) & V(f_3) \end{array}$$

← covariance matrix

iii) Eigen value & Eigen vectors. Find.

3D → 3 vectors

$\lambda_1, \lambda_2, \lambda_3$

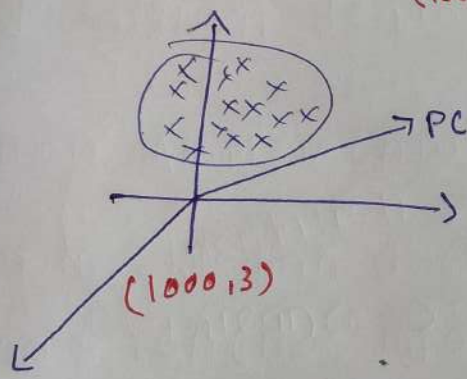
↓ PC1 ↓ PC2 ↓ PC3

PC1 = max. variance

How to transform points : 3D → 2D / 3D → 1D

For 1D → | f_1 | f_2 | f_3 | Y | ⟹ | PC1 | Y |

(1000, 4) (1000, 2)



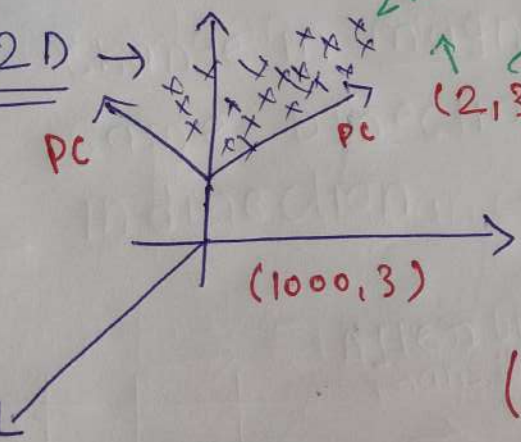
PC1 (1, 3) ivector 3D space

$U^T . X \Rightarrow (3, 1)$

(1000, 3) . (3, 1)

(1000, 1) = 1D

For 2D →



2vector

(2, 3) → 3D

$U^T . X \rightarrow (3, 2)$

(1000, 3) . (3, 2)

⇒ (1000, 2) = 2D

PCA (Principle Component Analysis):

Example-1 (Manual Method):

In [ ]:
```
# Import Libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

In [ ]:
```
# Dataset
data = {
    'X': [2.3, 3.0, 1.8, 4.2, 2.6, 3.7, 1.9, 4.5, 3.4, 2.1],
    'Y': [1.2, 1.5, 0.9, 2.1, 1.3, 1.8, 1.0, 2.3, 1.7, 1.1],
    'Z': [0.5, 0.7, 0.4, 0.9, 0.6, 0.8, 0.5, 1.0, 0.7, 0.5]
}

df = pd.DataFrame(data)
df
```

Out[ ]:

	X	Y	Z
0	2.3	1.2	0.5
1	3.0	1.5	0.7
2	1.8	0.9	0.4
3	4.2	2.1	0.9
4	2.6	1.3	0.6
5	3.7	1.8	0.8
6	1.9	1.0	0.5
7	4.5	2.3	1.0
8	3.4	1.7	0.7
9	2.1	1.1	0.5

In [ ]:

```
# Mean Centring / Scaling
scaler= StandardScaler()
df.iloc[:,0:3]=scaler.fit_transform(df.iloc[:,0:3])
df
```

Out[ ]:

	X	Y	Z
0	-0.708155	-0.643809	-0.862662
1	0.054473	0.022200	0.215666
2	-1.252890	-1.309818	-1.401826
3	1.361837	1.354219	1.293993
4	-0.381314	-0.421806	-0.323498
5	0.817102	0.688210	0.754829
6	-1.143943	-1.087815	-0.862662
7	1.688678	1.798225	1.833157
8	0.490261	0.466206	0.215666
9	-0.926049	-0.865812	-0.862662

In [ ]:
```
# Create an interactive 3D scatter plot with Plotly
fig = px.scatter_3d(df, x='X', y='Y', z='Z', title='3D Scatter Plot')
fig.show()
```

In [ ]:
```
# Step-2 (Find Covariance)
covraiance_matrix= np.cov([df.iloc[:,0],df.iloc[:,1],df.iloc[:,2]])
covraiance_matrix
```

Out[ ]:
```
array([[1.11111111, 1.10855046, 1.09648475],
       [1.10855046, 1.11111111, 1.09854359],
       [1.09648475, 1.09854359, 1.11111111]])
```

In [ ]:
```
# Step-3 (Find Eigen Values & Eigen Vectors)
eigen_values, eigen_vectors=np.linalg.eig(covraiance_matrix)
```

In [ ]:
```
eigen_values
```

Out[ ]:
```
array([3.31350540e+00, 2.46477813e-03, 1.73631574e-02])
```

In [ ]:
```
eigen_vectors
```

Out[ ]:
```
array([[-0.57781474, -0.67233823, -0.46270015],
       [-0.57817295,  0.73732757, -0.34937671],
       [-0.5760609 , -0.0656457 ,  0.81476652]])
```

In [ ]:
```
# Step-4 (Select No. of Principle Components)
pc= eigen_vectors[0:2]
```

In [ ]:
```
pc
```

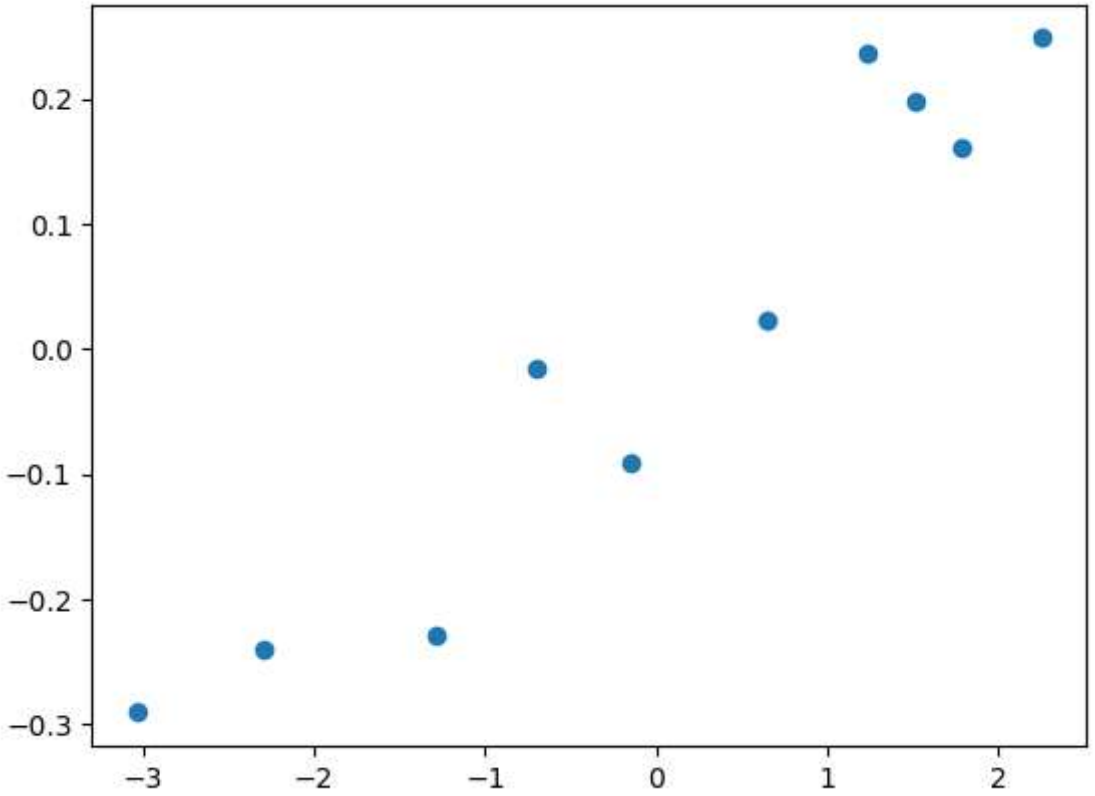
Out[ ]:
```
array([[-0.57781474, -0.67233823, -0.46270015],
       [-0.57817295,  0.73732757, -0.34937671]])
```

In [ ]:
```
# Step-5 (Transform Data 3D to 2D)
transformed_data= np.dot(df.iloc[:,0:3],pc.T)
new_df= pd.DataFrame(transformed_data,columns=['PC1','PC2'])
new_df
```

Out[ ]:

	PC1	PC2
0	1.241194	0.236132
1	-0.146190	-0.090475
2	2.253204	0.248387
3	-2.296113	-0.240966
4	0.653608	0.022479
5	-1.284103	-0.228710
6	1.791521	0.160715
7	-3.032960	-0.290929
8	-0.696517	-0.015057
9	1.516357	0.198424

In [ ]:
```
# Final 2D Plot
plt.scatter(new_df['PC1'],new_df['PC2'])
plt.show()
```



Example-2 (Manual Method):

In [ ]:
```
import pandas as pd
import random

data = {
    'Feature1': [2.3, 3.0, 1.8, 4.2, 2.6, 3.7, 1.9, 4.5, 3.4, 2.1],
    'Feature2': [1.2, 1.5, 0.9, 2.1, 1.3, 1.8, 1.0, 2.3, 1.7, 1.1],
    'Feature3': [0.5, 0.7, 0.4, 0.9, 0.6, 0.8, 0.5, 1.0, 0.7, 0.5],
```

```
    'Feature4': [3.1, 2.8, 3.4, 3.9, 3.0, 3.5, 3.2, 3.7, 2.9, 3.3],
    'Feature5': [2.0, 2.2, 2.1, 2.5, 2.3, 2.4, 2.0, 2.7, 2.2, 2.1]
}

# Create a target variable
target = ['Class A', 'Class A', 'Class B', 'Class B', 'Class A', 'Class A', 'Class B', 'Class A', 'Class B', 'Class B']
random.shuffle(target)

# Add the target variable to the DataFrame
df = pd.DataFrame(data)
df['Target'] = target
df
```

Out[ ]:

	Feature1	Feature2	Feature3	Feature4	Feature5	Target
0	2.3	1.2	0.5	3.1	2.0	Class B
1	3.0	1.5	0.7	2.8	2.2	Class B
2	1.8	0.9	0.4	3.4	2.1	Class A
3	4.2	2.1	0.9	3.9	2.5	Class A
4	2.6	1.3	0.6	3.0	2.3	Class B
5	3.7	1.8	0.8	3.5	2.4	Class B
6	1.9	1.0	0.5	3.2	2.0	Class A
7	4.5	2.3	1.0	3.7	2.7	Class A
8	3.4	1.7	0.7	2.9	2.2	Class A
9	2.1	1.1	0.5	3.3	2.1	Class B

In [ ]:

```
scaler= StandardScaler()
df.iloc[:,0:5]= scaler.fit_transform(df.iloc[:,0:5])
df
```

Out[ ]:

	Feature1	Feature2	Feature3	Feature4	Feature5	Target
0	-0.708155	-0.643809	-0.862662	-0.538816	-1.159347	Class B
1	0.054473	0.022200	0.215666	-1.436842	-0.231869	Class B
2	-1.252890	-1.309818	-1.401826	0.359211	-0.695608	Class A
3	1.361837	1.354219	1.293993	1.855921	1.159347	Class A
4	-0.381314	-0.421806	-0.323498	-0.838158	0.231869	Class B
5	0.817102	0.688210	0.754829	0.658553	0.695608	Class B
6	-1.143943	-1.087815	-0.862662	-0.239474	-1.159347	Class A
7	1.688678	1.798225	1.833157	1.257237	2.086825	Class A
8	0.490261	0.466206	0.215666	-1.137500	-0.231869	Class A
9	-0.926049	-0.865812	-0.862662	0.059868	-0.695608	Class B

In [ ]:
```
covraiance_matrix= np.cov([df.iloc[:,0],df.iloc[:,1],df.iloc[:,2],df.iloc[:,3],df.iloc[:,4]])
```

In [ ]:
```
covraiance_matrix
```

Out[ ]:
```
array([[1.11111111, 1.10855046, 1.09648475, 0.55803464, 1.01326552],
       [1.10855046, 1.11111111, 1.09854359, 0.56708164, 1.012356  ],
       [1.09648475, 1.09854359, 1.11111111, 0.55950087, 1.02790627],
       [0.55803464, 0.56708164, 0.55950087, 1.11111111, 0.69408304],
       [1.01326552, 1.012356  , 1.02790627, 0.69408304, 1.11111111]])
```

In [ ]:
```
eigen_values,eigen_vectors= np.linalg.eig(covraiance_matrix)
```

In [ ]:
```
eigen_values
```

Out[ ]:
```
array([4.68565034e+00, 7.43873889e-01, 1.08306526e-01, 2.30571180e-03,
       1.54190871e-02])
```

In [ ]:
```
eigen_vectors
```

Out[ ]:
```
array([[-0.47607277, -0.21981363,  0.28968257, -0.66425733, -0.44708265],
       [-0.47680085, -0.20912091,  0.31634636,  0.74026867, -0.28435309],
       [-0.47661819, -0.21448585,  0.1224803 , -0.09802908,  0.83798556],
       [-0.31549865,  0.92794147,  0.19599455, -0.01416529,  0.02776162],
       [-0.46830691,  0.02951033, -0.87326611,  0.03088929, -0.12755363]])
```

In [ ]:
```
# Index the eigenvalues in descending order
sorted_indices = np.argsort(eigen_values)[::-1]
# Sort the eigenvalues in descending order
sorted_eigenvalues = eigen_values[sorted_indices]
# sort the corresponding eigenvectors accordingly
sorted_eigenvectors = eigen_vectors[:, sorted_indices]
```

In [ ]:
```
(sorted_eigenvalues).round(2)
```

Out[ ]:
```
array([4.69, 0.74, 0.11, 0.02, 0.  ])
```

In [ ]:
```
sorted_eigenvectors
```

Out[ ]:
```
array([[-0.47607277, -0.21981363,  0.28968257, -0.44708265, -0.66425733],
       [-0.47680085, -0.20912091,  0.31634636, -0.28435309,  0.74026867],
       [-0.47661819, -0.21448585,  0.1224803 ,  0.83798556, -0.09802908],
       [-0.31549865,  0.92794147,  0.19599455,  0.02776162, -0.01416529],
       [-0.46830691,  0.02951033, -0.87326611, -0.12755363,  0.03088929]])
```

In [ ]:
```
pc= sorted_eigenvectors[:,0:2]
```

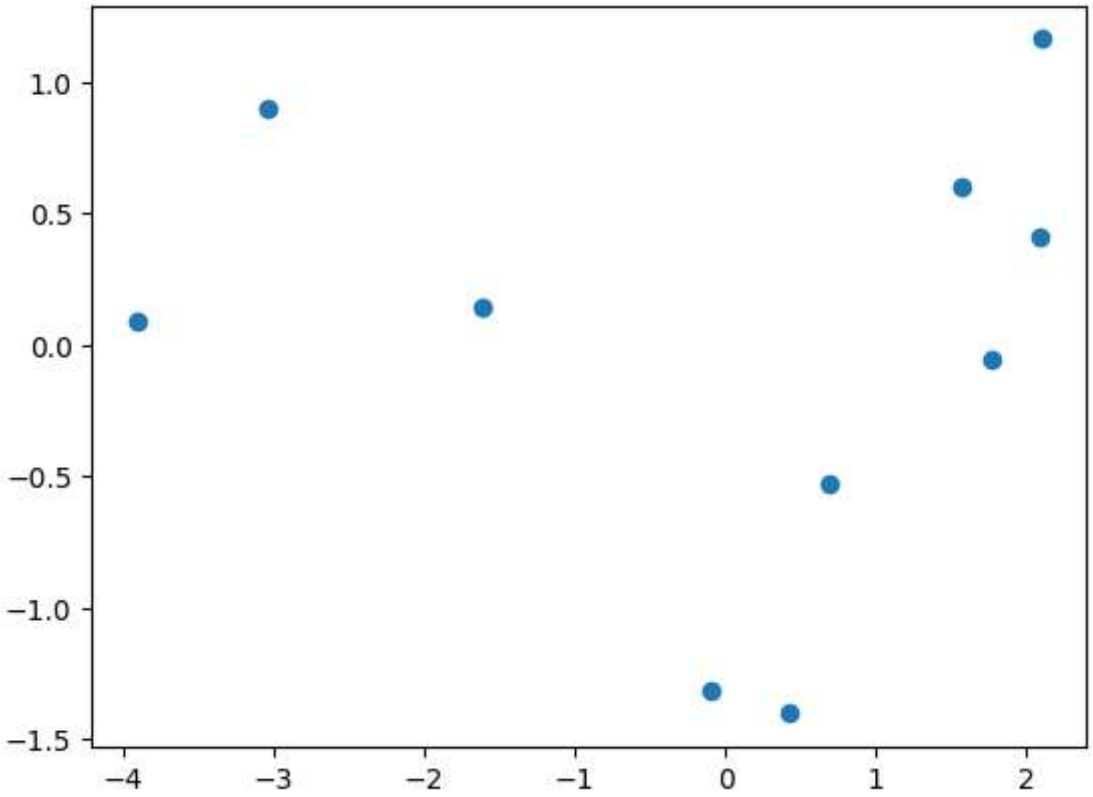
In [ ]:
```
pc
```

Out[ ]:
```
array([[-0.47607277, -0.21981363],
       [-0.47680085, -0.20912091],
       [-0.47661819, -0.21448585],
       [-0.31549865,  0.92794147],
       [-0.46830691,  0.02951033]])
```

In [ ]:
```
new_df= projected_data = np.dot(df.iloc[:, 0:5], pc[:, :2])  # Using the first 2 PCs
new_df = pd.DataFrame(projected_data, columns=['PC1', 'PC2'])
```

In [ ]:
```
new_df['Target']= df['Target']
```

In [ ]:
```
plt.scatter(new_df['PC1'],new_df['PC2'])
plt.show()
```



Example 3 (Manual Method):

In [ ]:
```
# Here we are using inbuilt dataset of scikit learn
from sklearn.datasets import load_breast_cancer

# instantiating
cancer = load_breast_cancer(as_frame=True)
# creating dataframe
df = cancer.frame
```

```
# checking shape
print('Original Dataframe shape :',df.shape)

# Input features
X = df[cancer['feature_names']]
print('Inputs Dataframe shape :', X.shape)
```

```
Original Dataframe shape : (569, 31)
Inputs Dataframe shape : (569, 30)
```

In [ ]:
```
y= df['target']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
model= SVC()
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

Out[ ]:
```
0.9473684210526315
```

In [ ]:
```
scaler= StandardScaler()
X.iloc[:,:30]=scaler.fit_transform(X.iloc[:,:30])
```

```
C:\Users\Shorya Sharma\AppData\Local\Temp\ipykernel_11112\2990642078.py:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [ ]:
```
covariance_matrix= X.cov()
```

In [ ]:
```
eigen_values,eigen_vectors= np.linalg.eig(covariance_matrix)
```

In [ ]:
```
(eigen_values).round(1)
```

Out[ ]:
```
array([13.3,  5.7,  2.8,  2. ,  1.7,  1.2,  0.7,  0.5,  0.4,  0.4,  0.3,
       0.3,  0.2,  0.2,  0.1,  0.1,  0.1,  0.1,  0. ,  0. ,  0. ,  0. ,
       0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ])
```

In [ ]:
```
idx= np.argsort(eigen_values)[::-1]
sorted_eigenvalues= eigen_values[idx]
sorted_eigenvectors= eigen_vectors[:,idx]
```

In [ ]:
```
(sorted_eigenvalues).round(1)
```

Out[ ]:
```
array([13.3,  5.7,  2.8,  2. ,  1.7,  1.2,  0.7,  0.5,  0.4,  0.4,  0.3,
       0.3,  0.2,  0.2,  0.1,  0.1,  0.1,  0.1,  0. ,  0. ,  0. ,  0. ,
       0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ])
```

In [ ]:
```
pc= sorted_eigenvectors[:,0:5]
```

In [ ]:
```
transformed_data= np.dot(df.iloc[:,0:-1],pc[:,0:5])
new_df= pd.DataFrame(transformed_data,columns=['PC1','PC2','PC3','PC4','PC5'])
new_df
```

Out[ ]:

	PC1	PC2	PC3	PC4	PC5
0	793.364674	-772.386400	28.693570	114.456637	61.498704
1	831.376304	-819.573038	21.695304	113.079259	46.296461
2	751.111800	-737.924156	26.218798	98.643829	44.035575
3	271.973741	-262.394926	4.912013	14.387632	15.271273
4	741.248858	-730.596479	30.886047	110.336027	38.163621
...	...	...	...	...	...
564	904.399203	-887.916357	44.709909	128.359133	58.179018
565	772.937544	-758.802048	29.407552	90.168986	46.543022
566	522.486870	-512.141833	15.075822	48.895637	27.853743
567	801.865190	-785.872631	23.540833	91.031488	46.660824
568	139.519596	-132.514150	3.779765	-12.267499	10.284296

569 rows × 5 columns

In [ ]:
```
X= new_df
y= df['target']
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [ ]:
```
model= SVC()
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

Out[ ]:
```
0.9473684210526315
```