PCA Numerical

By: Shrijak Dahal

Code Link: https://github.com/shrijak/PCA_Numerical/blob/main/PCA_Numerical.ipynb

Step 1:1

➢ Inserting data

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
X1 = np.array((2.5,0.5,2.2,1.9,3.1,2.3,2,1,1.5,1.2))
X2 = np.array((2.4,0.7,2.9,2.2,3,2.7,1.6,1.1,1.6,0.9))
X_unorm = np.array((X1,X2))
X_unorm
```

```
array([[2.5, 0.5, 2.2, 1.9, 3.1, 2.3, 2. , 1. , 1.5, 1.2],
       [2.4, 0.7, 2.9, 2.2, 3. , 2.7, 1.6, 1.1, 1.6, 0.9]])
```

Step 1:2

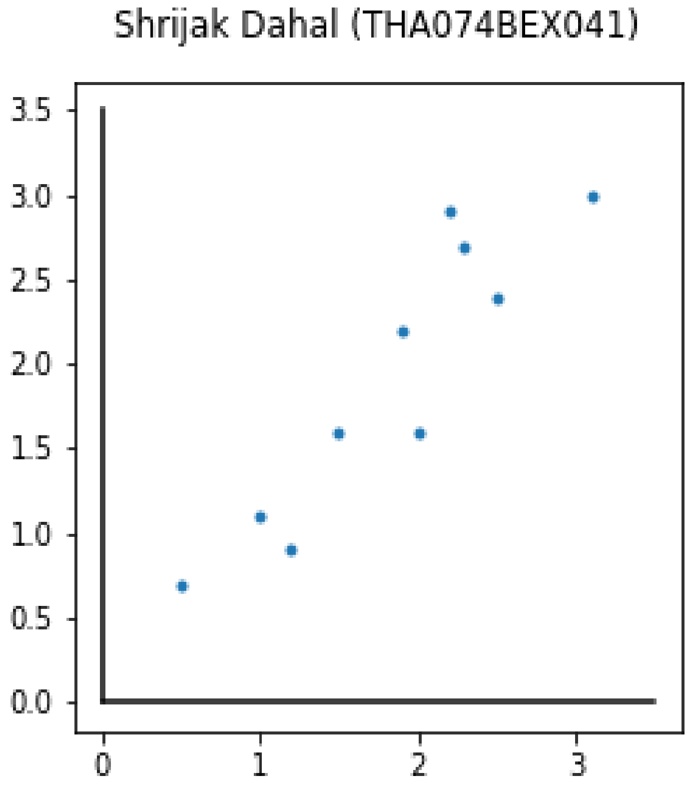
- Subtracting mean from each of the dimensions

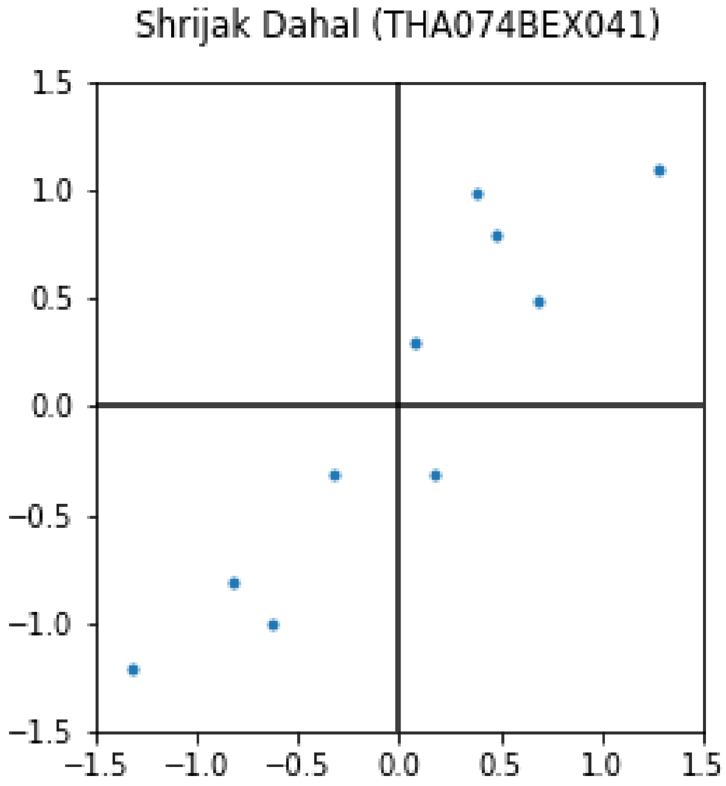
```
X1_norm = X1 - X1.mean()
X2_norm = X2 - X2.mean()
X = np.array((X1_norm,X2_norm))
X
```

```
array([[ 0.68, -1.32,  0.38,  0.08,  1.28,  0.48,  0.18, -0.82, -0.32,
        -0.62],
       [ 0.49, -1.21,  0.99,  0.29,  1.09,  0.79, -0.31, -0.81, -0.31,
        -1.01]])
```

Step 1:2

➢ Visualization of data (Raw data and Zero Mean data)





Step 2

- Calculation of covariance matrix of X

$$\mathbf{S_X} \equiv \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \quad \text{where} \quad \mathbf{X} = \begin{bmatrix} \mathbf{x_1} \\ \vdots \\ \mathbf{x_m} \end{bmatrix}$$

Step 2

- Calculation of covariance matrix of X

```
Sx = (1 / (X.shape[1] - 1)) * np.matmul(X,np.transpose(X))
Sx
```

```
array([[0.60177778, 0.60422222],
       [0.60422222, 0.71655556]])
```

Step 3:1

- Calculation of eigenvalue of covariance matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$m = \frac{a + d}{2}$$

$$p = ad \ - bc$$

$$\lambda_1, \lambda_2 = \mathrm{m} \ \pm \ \sqrt{m^2 \ - p}$$

Step 3:1

- Calculation of eigenvalue of covariance matrix

```
# Eigen value Calculation

m = (Sx[0][0]+Sx[1][1])/2
p = Sx[0][0]*Sx[1][1] - Sx[0][1]*Sx[1][0]
lambda1 = m + np.sqrt(m**2 - p)
lambda2 = m - np.sqrt(m**2 - p)
print("lambda1: "+str(round(lambda1,2))+"\nlambda2: "+str(round(lambda2,2)))
```

```
lambda1: 1.27
lambda2: 0.05
```

Step 3:2

- Calculation of eigenvector of covariance matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}$$

- Choosing first equation,

$$x = \frac{b}{\lambda - a} \times y$$

Step 3:2

- Calculation of eigenvector of covariance matrix

- For λ =λ_1 , and y_1 = 1

$$x_1 = \frac{b}{\lambda_1 - a} \times 1$$

- Similarly, For λ =λ_2 , and y_2 = 1

$$x_2 = \frac{b}{\lambda_2 - a} \times 1$$

Step 3:2

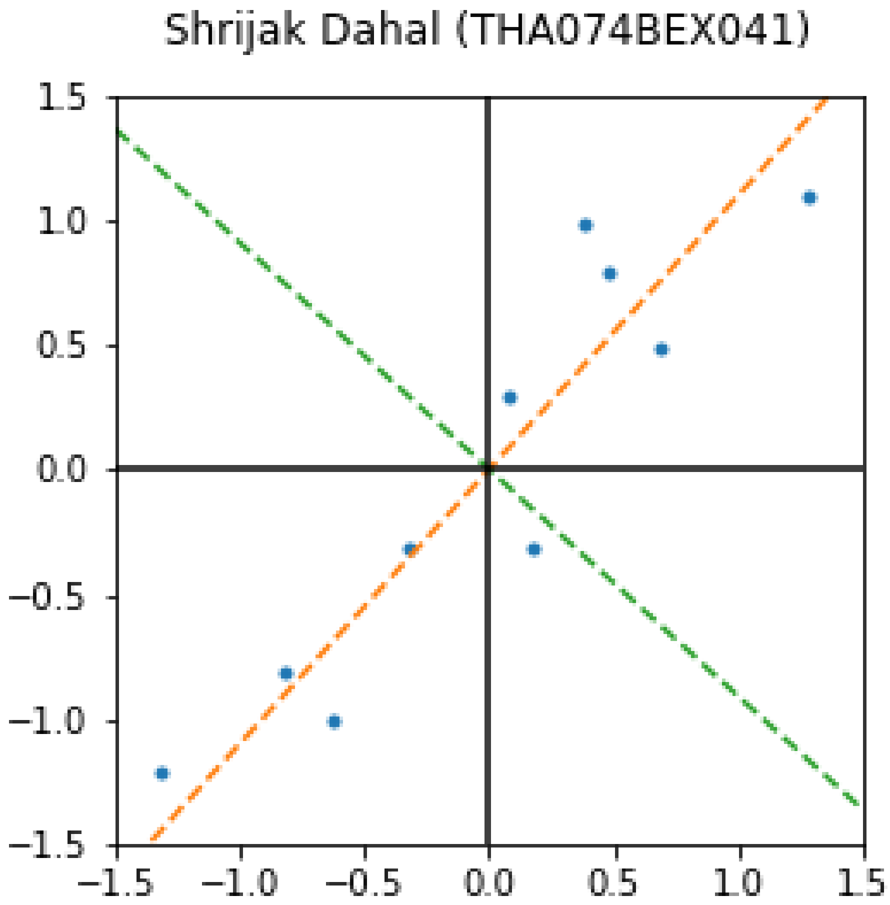
- Calculation of eigenvector of covariance matrix

```
# Eigen Vector Calculation

v1 = np.array((Sx[0][1] / (lambda1-Sx[0][0]),1))
v2 = np.array((Sx[0][1] / (lambda2-Sx[0][0]),1))
print("eigen vector v1:" )
print(v1)
print("\neigen vector v2:" )
print(v2)
```

```
eigen vector v1:
[0.90952068 1.        ]

eigen vector v2:
[-1.09948022  1.        ]
```

Step 3:2

- Visualization of eigenvector lines



Step 4

➢ Variance explained by two components

$$\frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{m} \lambda_i} = \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_r}{\lambda_1 + \lambda_2 + \ldots + \lambda_p + \ldots + \lambda_m}$$

Step 4

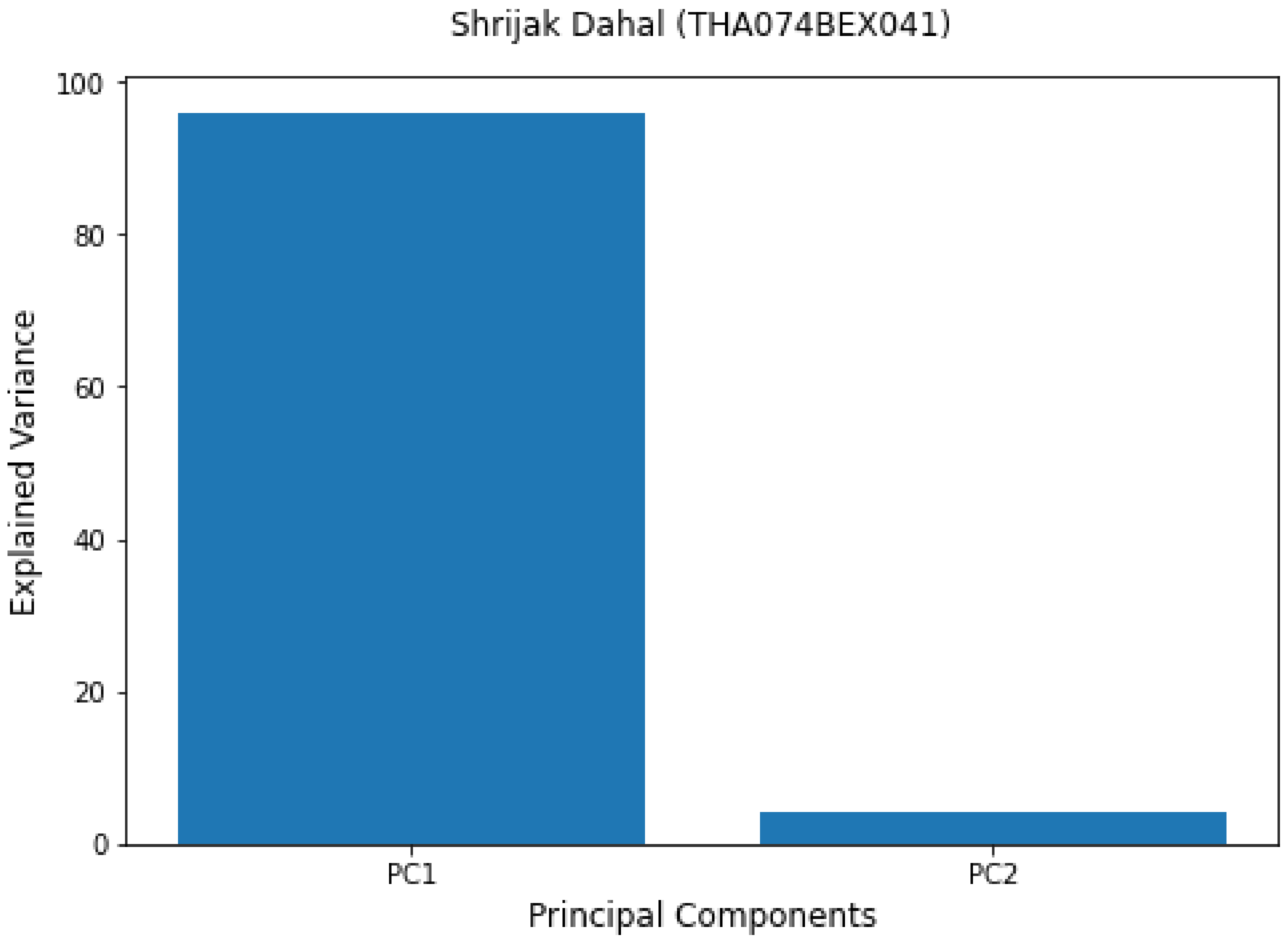
➢ Variance explained by two components

```
k = lambda1 / (lambda1 + lambda2)
print(str(round(k*100,2))+'% variance is explained by v1')
print(str(round((1-k)*100,2))+'% variance is explained by v2')
```

```
96.04% variance is explained by v1
3.96% variance is explained by v2
```

Step 4

- Visualization of Variance explained by two components



Step 5:1

- Choosing both components and finding value of Y (No PCA)

```
P = np.array((v1,v2))    # (0.6718*v1,-0.7406*v2)
Y = np.matmul(P,X)
Y
```

```
array([[ 1.10847406, -2.4105673 ,  1.33561786,  0.36276165,  2.25418647,
         1.22656993, -0.14628628, -1.55580696, -0.60104662, -1.57390282],
       [-0.25764655,  0.2413139 ,  0.57219752,  0.20204158, -0.31733469,
         0.26224949, -0.50790644,  0.09157378,  0.04183367, -0.32832226]])
```

Step 5:1

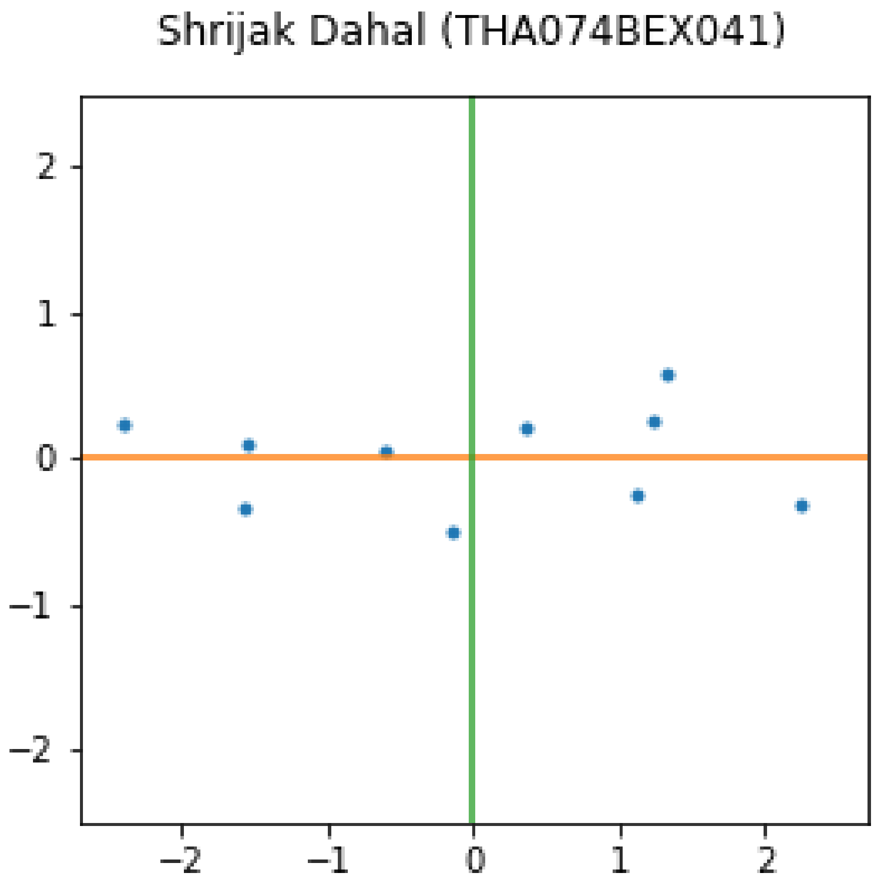
- Calculation of Co-variance matrix of Y

```
Sy = (1 / (Y.shape[1] - 1)) * np.matmul(Y,np.transpose(Y))
Sy
```

```
array([[2.31346812e+00, 1.90865418e-16],
       [1.90865418e-16, 1.15357923e-01]])
```

Step 5:1

- ➢ Visualization of Choosing both components and finding value of Y (No PCA)



Step 5:2

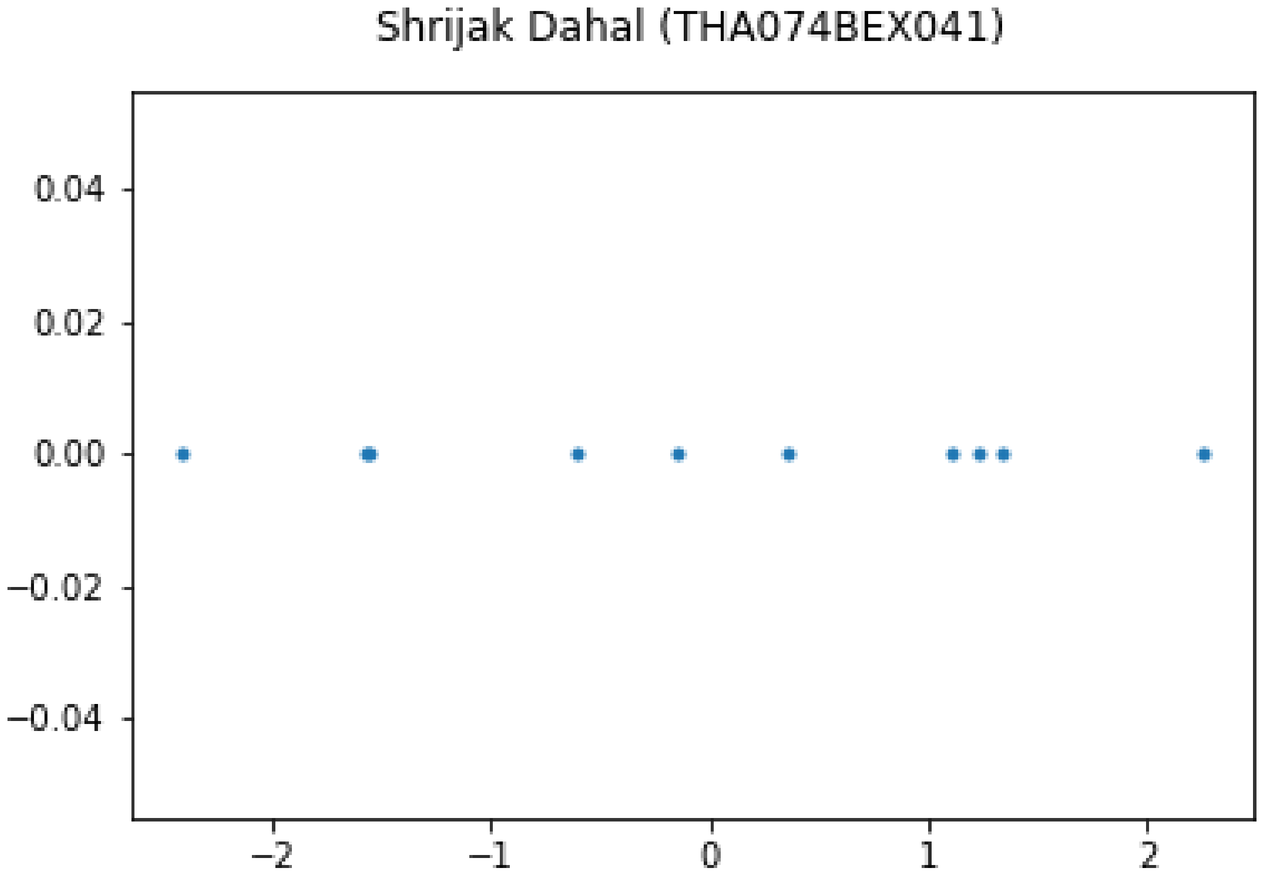
- Choosing principal component v1 and finding value of Y

```
P_pca = v1
Y_pca = np.matmul(P_pca,X)
Y_pca
```

```
array([ 1.10847406, -2.4105673 ,  1.33561786,  0.36276165,  2.25418647,
        1.22656993, -0.14628628, -1.55580696, -0.60104662, -1.57390282])
```

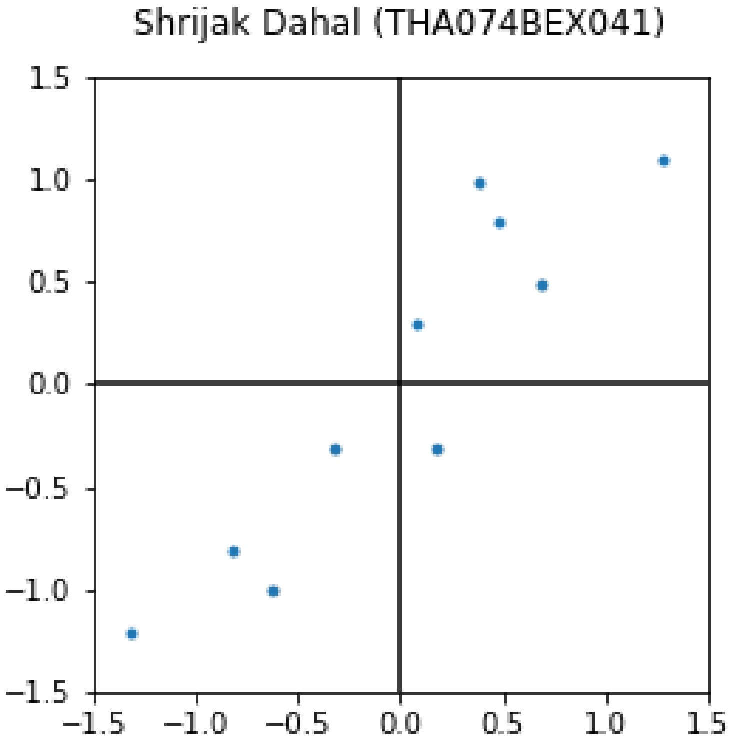
Step 5:2

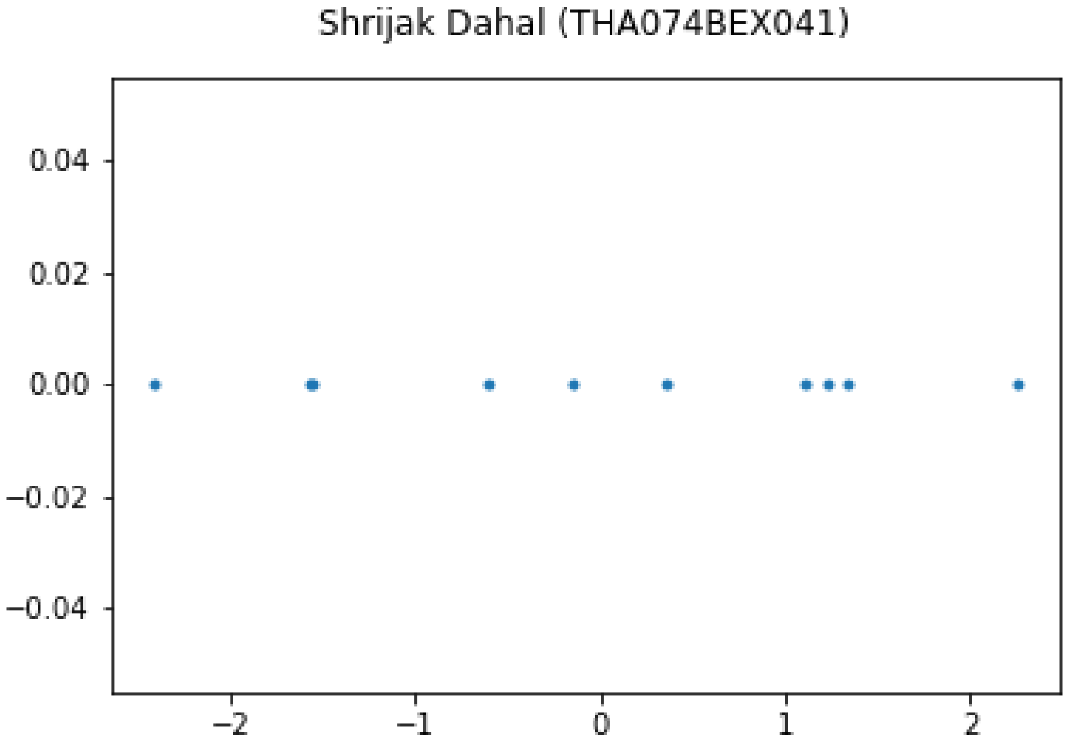
- Visualization of Choosing principal component v1 and finding value of Y



Conclusion

- So finally using PCA we were able to convert 2D data to 1D with 96% variance explained





Thank You