Multivariate Analysis

Assignment 8

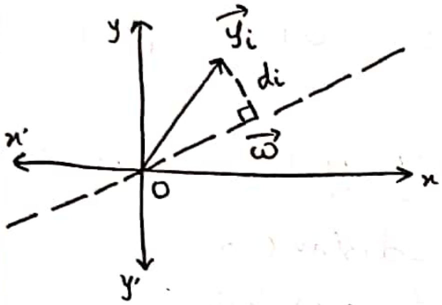
1) Y : centered data matrix of dimension $n \times p$ (let)

y_i : $p \times 1$ vector representing the i^{th} datapoint

if ω is a unit vector :

Objective : $\underset{\omega}{\text{minimize}} \sum_{i=1}^{n}$ (squared orthogonal distance of y_i from ω)

such that $\|\omega\| = 1$ —— (I)



: Based on the figure

$$d_i^2 = \|y_i\|^2 - \|y_i \cdot \vec{w}\|^2$$
$$= y_i \cdot y_i^T - (y_i^T \cdot \omega)^2$$
$$= y_i^T \cdot y_i - \omega^T \cdot y_i \cdot y_i^T \omega$$

Now, $\underset{\|\omega\|=1}{\text{argmin}_\omega} \sum_{i=1}^{n} d_i^2 = \underset{\|\omega\|=1}{\text{argmin}_\omega} \sum_{i=1}^{n} (y_i^T \cdot y_i - \omega^T \cdot y_i \cdot y_i^T \omega)$

$= \underset{\|\omega\|=1}{\text{argmax}_\omega} \sum_{i=1}^{n} (\omega^T y_i y_i^T \omega) = \underset{\|\omega\|=1}{\text{argmax}_\omega} \omega^T \left(\sum_{i=1}^{n} y_i^T y_i\right) \omega$

[Y is centered $\rightarrow \bar{Y} = 0$] $= \underset{\|\omega\|=1}{\text{argmax}_\omega} \omega^T S \omega$ {S being the data covariance matrix}

$= \text{argmax}_\omega \; \omega^T S \omega + \lambda(1 - \omega^T \omega)$ [Use of lagrangian]

let $F(\omega) = \omega^T S \omega + \lambda(1 - \omega^T \omega)$

$\frac{\partial F}{\partial \omega} = 0 \Rightarrow 2S\omega - 2\lambda\omega = 0 \Rightarrow (S - I\lambda)\,\omega = 0$

$\Rightarrow (S - \lambda I)\omega = 0$ [Eigenvalue problem]

And $S\omega = \lambda\omega$

$\therefore \omega^T S \omega = \omega^T \lambda \omega = \lambda \omega^T \omega = \lambda$.

To maximize $\omega^T S \omega$ or to achieve (I), we consider the largest eigenvalue of S as λ and choose the corresponding eigenvector as ω. Hence, the direction of the first principal component is given by the eigenvector corresponding to the largest eigenvalue of S. [Proved]

2) Proposition: Principal Components are scale invariant.

We construct a counterexample to disprove the above Proposition. Consider the case presented below.

- After scaling the data; we see that the directions of the principal components change significantly.
  - this is illustrated both by
    - computing the eigenvectors
    - plotting the data along with the principal components.

A scale change in a variable leads to a change in the shape of the swarm of points in the dataset.

Based on the illustrations within the presented counterexample, the proposition is disproved. Therefore, principal components are not scale invariant. [Proved].

Problem 2

April 21, 2021

```
In [1]: # Simple example to illustrate that principal components are not scale invariant

In [2]: import numpy as np
        import matplotlib.pyplot as plt

In [3]: size = 20
        np.random.seed(42)
        x = [np.random.uniform(-2, 2) for index in range(size)]
        y = [np.random.uniform(-5, 5) for index in range(size)]

        points = [[x[i], y[i]] for i in range(len(x))]
        points = np.asarray(points)

        mean = np.mean(points, axis = 0)
        covariance_matrix = np.cov(points.T)
        eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)

        order = np.argsort(-1 * eigen_values)

        print("Principal Eigen Value:", eigen_values[order[0]])
        print("Principal Eigen Vector:", eigen_vectors[:, order[0]])
        print()

        print("Secondary Eigen Value:", eigen_values[order[1]])
        print("Secondary Eigen Vector:", eigen_vectors[:, order[1]])
```

```
Principal Eigen Value: 9.069725862741496
Principal Eigen Vector: [ 0.26939593 -0.96302951]

Secondary Eigen Value: 0.9235364068391636
Secondary Eigen Vector: [-0.96302951 -0.26939593]
```

1 *Principal Components of the raw data*

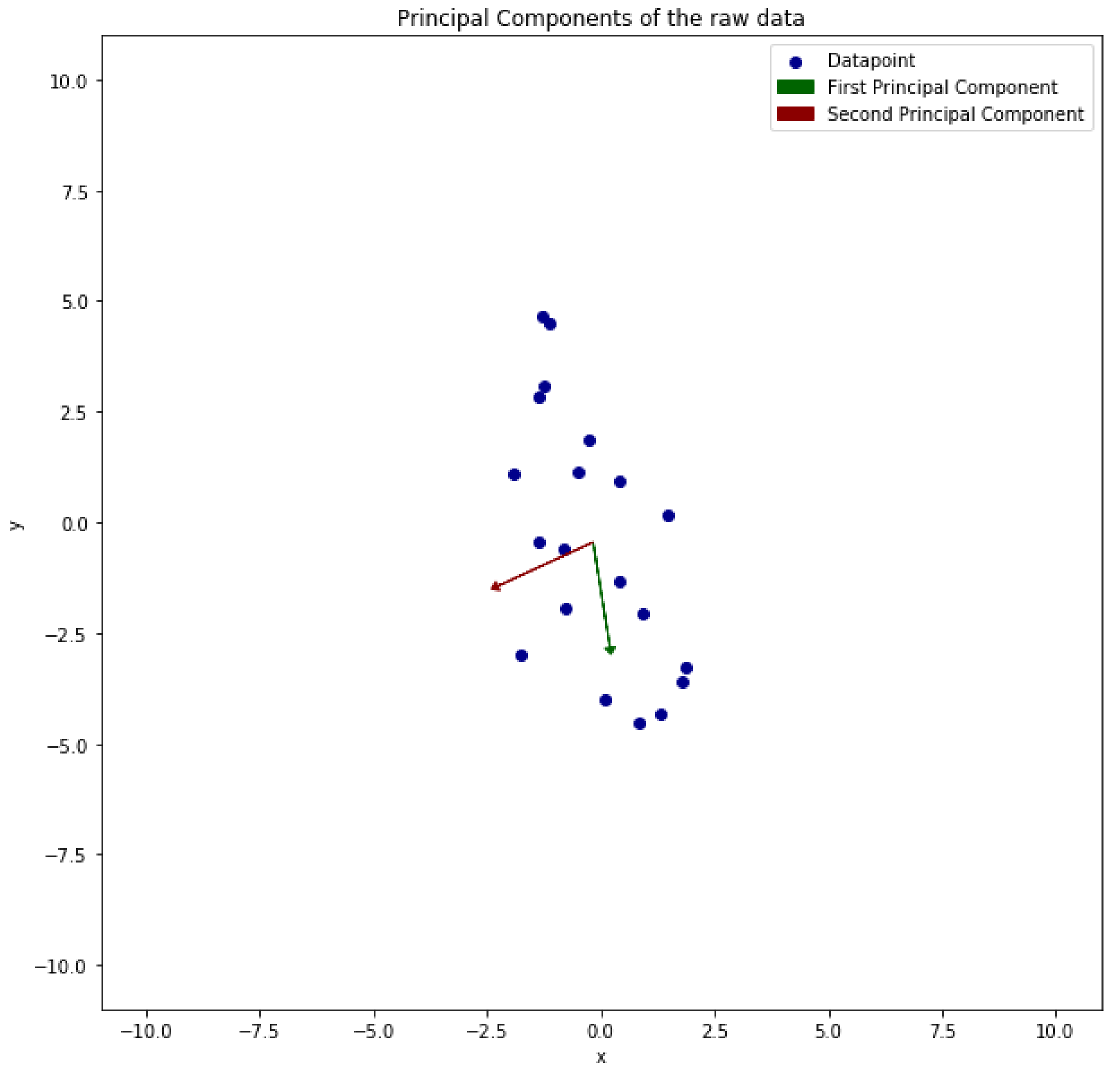
```
In [4]: plt.figure(figsize = (10, 10))
        fig = plt.subplot(111)
```

```
fig.set_title('Principal Components of the raw data')
fig.set_xlabel('x')
fig.set_ylabel('y')

scale = 0.65
length_scale = 2

fig.set_xlim(-11, 11)
fig.set_ylim(-11, 11)
data = fig.scatter(points[:, 0], points[:, 1], color = 'darkblue', label = '')
arrow_a = fig.arrow(mean[0],
                    mean[1],
                    mean[0] + eigen_vectors[:, order[0]][0] * length_scale,
                    mean[1] + eigen_vectors[:, order[0]][1] * length_scale,
                    color = 'darkgreen',
                    head_width = 0.35 * scale,
                    head_length = 0.25 * scale,
                    label = '')
arrow_b = fig.arrow(mean[0],
                    mean[1],
                    mean[0] + eigen_vectors[:, order[1]][0] * length_scale,
                    mean[1] + eigen_vectors[:, order[1]][1] * length_scale,
                    color = 'darkred',
                    head_width = 0.35 * scale,
                    head_length = 0.25 * scale,
                    label = '')

fig.legend([data, arrow_a, arrow_b],
           ['Datapoint', 'First Principal Component', 'Second Principal Component'])
plt.show()
```




```
In [5]: # Scale change
        change = 5
        new_x = [change * unit for unit in x]

        points = [[new_x[i], y[i]] for i in range(len(x))]
        points = np.asarray(points)

        mean = np.mean(points, axis = 0)
        covariance_matrix = np.cov(points.T)
        eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)

        order = np.argsort(-1 * eigen_values)

        print("Principal Eigen Value:", eigen_values[order[0]])
```

```
print("Principal Eigen Vector:", eigen_vectors[:, order[0]])
print()

print("Secondary Eigen Value:", eigen_values[order[1]])
print("Secondary Eigen Vector:", eigen_vectors[:, order[1]])
```

```
Principal Eigen Value: 41.27338428038719
Principal Eigen Vector: [ 0.95180951 -0.30668983]

Secondary Eigen Value: 5.0736220086711175
Secondary Eigen Vector: [0.30668983 0.95180951]
```

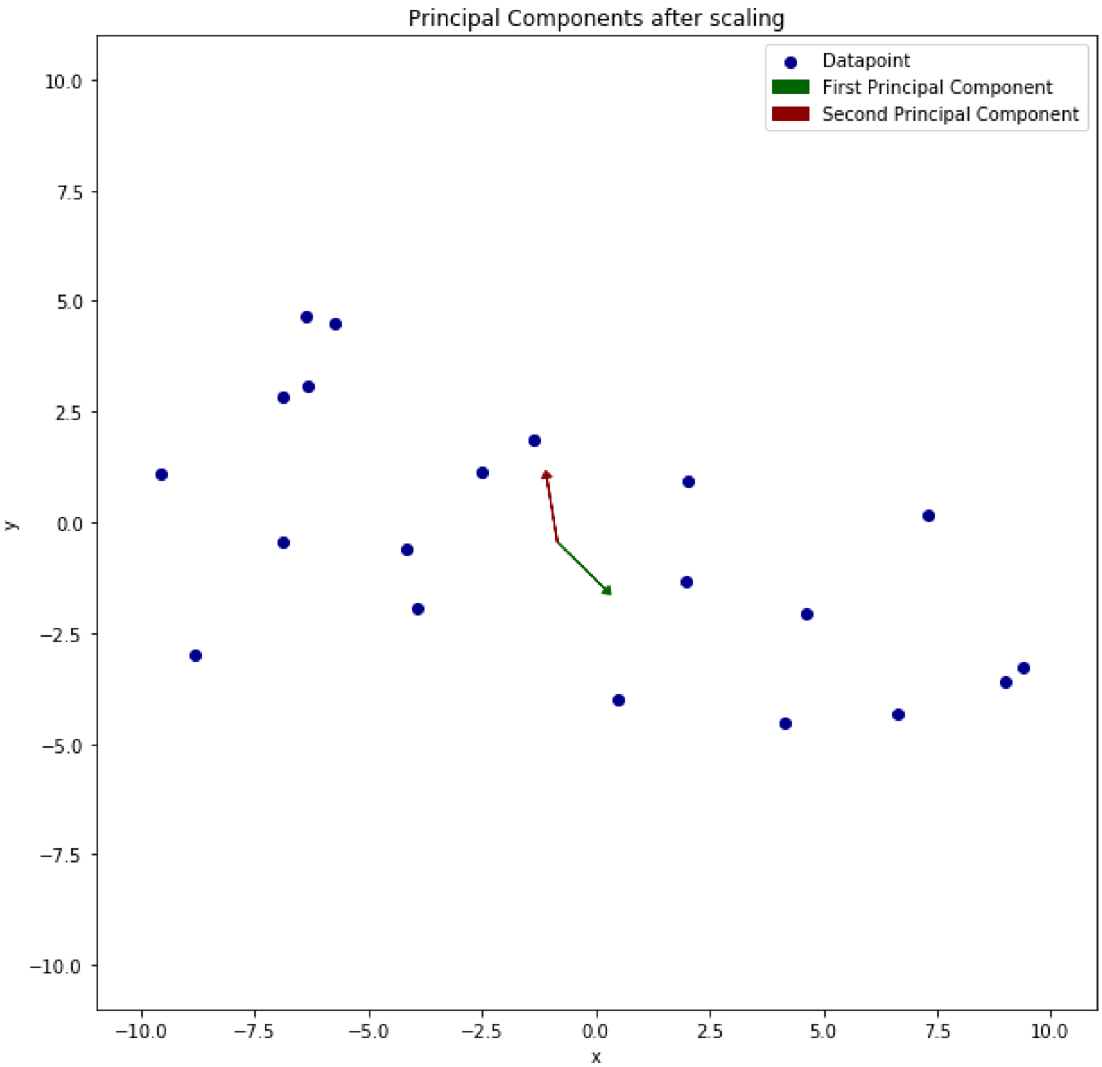
2 *Principal Components after scaling*

In [6]:
```
plt.figure(figsize = (10, 10))
fig = plt.subplot(111)
fig.set_title('Principal Components after scaling')
fig.set_xlabel('x')
fig.set_ylabel('y')

scale = 0.65
length_scale = 2

fig.set_xlim(-11, 11)
fig.set_ylim(-11, 11)
data = fig.scatter(points[:, 0], points[:, 1], color = 'darkblue', label = '')
arrow_a = fig.arrow(mean[0],
                    mean[1],
                    mean[0] + eigen_vectors[:, order[0]][0] * length_scale,
                    mean[1] + eigen_vectors[:, order[0]][1] * length_scale,
                    color = 'darkgreen',
                    head_width = 0.35 * scale,
                    head_length = 0.25 * scale,
                    label = '')
arrow_b = fig.arrow(mean[0],
                    mean[1],
                    mean[0] + eigen_vectors[:, order[1]][0] * length_scale,
                    mean[1] + eigen_vectors[:, order[1]][1] * length_scale,
                    color = 'darkred',
                    head_width = 0.35 * scale,
                    head_length = 0.25 * scale,
                    label = '')

fig.legend([data, arrow_a, arrow_b],
           ['Datapoint', 'First Principal Component', 'Second Principal Component'])
plt.show()
```




```
In [7]: # ^_^ Thank You
```

3) y_p is uncorrelated with y_i $\forall$ $i \in \{1, 2, \dots, p-1\}$ [Given]

$\rightarrow \forall i \in \{1, 2, \dots, p-1\}$; $cov(y_p, y_i) = 0$

$\rightarrow \forall i \in \{1, 2, \dots, p-1\}$; $s_{ip} = s_{pi} = 0$ — (i)

let $a = [0\ 0\ \dots\ 0\ 1]^T_{1\times p}$

$(S - \lambda I)\omega = 0$
- eigenvalue problem with solution: λ (eigenvalue), ω (eigenvector)

Now, $Sa = [s_{1p}\ \ s_{2p}\ \dots\ s_{pp}]^T$

$= [0\ \ 0\ \dots\ 0\ s_{pp}]^T$ (by (i))

$= [0\ \ 0\ \dots\ 0\ s_p^2]^T = s_p^2\,[0\ 0\ \dots\ 0\ 1]^T = s_p^2\, a$

$\therefore Sa - s_p^2 a = 0 \Rightarrow (S - s_p^2 I)a = 0$

$\therefore s_p^2$ is an eigenvalue of S with corresponding eigenvector as a.

4) A rotational transform preserves all mutual distances.

Let x_i, y_i be any two random non-zero vectors in the coordinate space.

$z_i = Ay_i$ and $w_i = Ax_i$. Now;

- $z_i^T z_i = y_i^T A^T A y_i = y_i^T I y_i = y_i^T y_i$ [Preserves distance from origin]
- $z_i^T w_i = y_i^T A^T A x_i = y_i^T I x_i = y_i^T x_i$ [Preserves dot products]

— As $A^T A = I$ (A is orthogonal)

$\therefore$ the transformation preserves mutual distances and dot products.

$\rightarrow$ A is a rotation matrix.

- $\hat{u}_k$: unit vector with the k^{th} component as 1.

$A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_p^T \end{bmatrix}$ (let) $\quad \hat{u}_k = Ag_k$ for some g_k

$\Rightarrow A^T\hat{u}_k = A^T A g_k \Rightarrow A^T \hat{u}_k = g_k$

$\Rightarrow g_k = A^T\hat{u}_k = [a_1\ a_2\ \dots\ a_p]\,\hat{u}_k = a_k$

$\hat{u}_k$ (a unit vector) in the z-space is equivalent to a_k in the y-space. Thus, A rotates the axes to align with those of the principal components. [Proved]

5>

1. λ: eigenvalue of A with corresponding eigenvector x.
$(A-bI)x = Ax - bx = \lambda x - bx = (\lambda - b)x$ [Proved]
As $Ax = \lambda x$ by definition of eigenvalue & eigenvector.

2. $\underline{\lambda_i}$: i^{th} eigenvalue of cA with corresponding eigenvector $\underline{x_i}$.
Also, let λ_i : i^{th} eigenvalue of A with corresponding eigenvector x_i.
$\therefore (cA - \underline{\lambda_i} I)\underline{x_i} = 0$
& $(A - \lambda_i I)x_i = 0 \Rightarrow (cA - c\lambda_i I)x_i = 0$
$\Rightarrow (cA - (c\lambda_i)I)x_i = 0$ — (i)
$\therefore$ An eigenvalue of A is also an eigenvector of cA with a different eigenvalue.
Since (i) holds WLG $\forall i$, A & cA have the exact same eigenvectors. And if $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ is the eigenvalue set of A, then $\{c\lambda_1, c\lambda_2, \dots, c\lambda_n\}$ is the eigenvalue set of cA. [As λ_i is an eigenvalue of $A \Rightarrow c\lambda_i$ is an eigenvalue of cA $\forall\ i = 1(1)n$]

3. Assign $R_k \leftarrow kR - (k-1)I$

4. Let λ_k be an eigenvalue of R_k with corresponding eigenvector x_k.
$\therefore (R_k - \lambda_k I)x_k = 0 \Rightarrow (kR - (k-1)I - \lambda_k I)x_k = 0$
$\Rightarrow (kR - (k-1+\lambda_k)I)x_k = 0$ — (i)
if λ, x are an eigenvalue, eigenvector pair of R:
$(R - \lambda I)x = 0 \Rightarrow (kR - k\lambda I)x = 0$ — (ii)
(i) & (ii) represent solutions to the same eigenvalue problem for kR.
$[(A - bI)x = (\lambda - b)x]$ $\therefore (kR - (k-1+\lambda_k)I)x = (k\lambda - (k-1+\lambda_k))x$
Now, $(k\lambda - (k-1+\lambda_k))x = 0$
$\Rightarrow k\lambda - (k-1+\lambda_k) = 0$
$\Rightarrow \lambda = \dfrac{k-1+\lambda_k}{k} = 1 + \dfrac{\lambda_k - 1}{k}$ {Proved}

Problems based on datasets

April 19, 2021

```
In [1]: import numpy as np

In [2]: def compute_contributions(matrix):
          eigenvalues, _ = np.linalg.eig(matrix)
          arrange = np.argsort(-1 * eigenvalues)
          ordered_values = eigenvalues[arrange]

          N = np.shape(ordered_values)[0]
          eigenvalue_sum = np.sum(ordered_values)
          percentages = ordered_values * (100 / eigenvalue_sum)

          average_contribution = np.mean(ordered_values)
          count = 0

          print('Number of Components: ' + str(N))
          print('Percentage Contributions: ')

          for index in range(N):
            print('    ' + str(index + 1) + '. ' +
                  str(np.round(percentages[index], 4)) + ' %')
            if ordered_values[index] > average_contribution:
              count += 1

          print('Number of components to be retained = ' + str(count))

        def get_results(data):
          # Computing S and R
          S = np.cov(data, rowvar = False)
          R = np.corrcoef(data, rowvar = False)

          print('- Using S: ')
          compute_contributions(S)

          print()
          print('- Using R: ')
          compute_contributions(R)
```

1 First Dataset

```
In [3]: data = [[51, 36, 50, 35, 42],
                [27, 20, 26, 17, 27],
                [37, 22, 41, 37, 30],
                [42, 36, 32, 34, 27],
                [27, 18, 33, 14, 29],
                [43, 32, 43, 35, 40],
                [41, 22, 36, 25, 38],
                [38, 21, 31, 20, 16],
                [36, 23, 27, 25, 28],
                [26, 31, 31, 32, 36],
                [29, 20, 25, 26, 25]]

        data = np.asmatrix(data)
        get_results(data)
```

```
- Using S:
Number of Components: 5
Percentage Contributions:
   1. 68.413 %
   2. 12.3223 %
   3. 11.633 %
   4. 5.1102 %
   5. 2.5215 %
Number of components to be retained = 1

- Using R:
Number of Components: 5
Percentage Contributions:
   1. 68.3299 %
   2. 12.2886 %
   3. 11.4455 %
   4. 5.4242 %
   5. 2.5118 %
Number of components to be retained = 1
```

2 Second Dataset

```
In [4]: data = [[60, 69, 62, 97, 69, 98],
                [56, 53, 84, 103, 78, 107],
                [80, 69, 76, 66, 99, 130],
                [55, 80, 90, 80, 85, 114],
                [62, 75, 68, 116, 130, 91],
                [74, 64, 70, 109, 101, 103],
                [64, 71, 66, 77, 102, 130],
                [73, 70, 64, 115, 110, 109],
```

```
                 [68, 67, 75, 76, 85, 119],
                 [69, 82, 74, 72, 133, 127],
                 [60, 67, 61, 130, 134, 121],
                 [70, 74, 78, 150, 158, 100],
                 [66, 74, 78, 150, 131, 142],
                 [83, 70, 74, 99, 98, 105],
                 [68, 66, 90, 119, 85, 109],
                 [78, 63, 75, 164, 98, 138],
                 [77, 68, 74, 144, 71, 153],
                 [66, 77, 68, 77, 82, 89],
                 [70, 70, 72, 114, 93, 122],
                 [75, 65, 71, 77, 70, 109]]

        data = np.asmatrix(data)
        get_results(data)
```

```
- Using S:
Number of Components: 6
Percentage Contributions:
   1. 54.4487 %
   2. 25.3703 %
   3. 12.8591 %
   4. 3.4688 %
   5. 2.4927 %
   6. 1.3604 %
Number of components to be retained = 2

- Using R:
Number of Components: 6
Percentage Contributions:
   1. 25.9397 %
   2. 24.7104 %
   3. 17.7156 %
   4. 16.0837 %
   5. 10.4815 %
   6. 5.0691 %
Number of components to be retained = 3
```

3 Third Dataset

```
In [5]: data = [[5.7, 4.67, 17.6, 1.50, 0.104, 1.50, 1.88, 5.15, 8.40, 7.5, 0.11],
                 [5.5, 4.67, 13.4, 1.65, 0.245, 1.32, 2.24, 5.75, 4.50, 7.1, 0.11],
                 [4.6, 2.70, 20.3, 0.90, 0.097, 0.89, 1.28, 4.35, 1.20, 2.3, 0.11],
                 [5.7, 3.49, 22.3, 1.75, 0.174, 1.50, 2.24, 7.55, 2.75, 4.0, 0.12],
                 [5.6, 3.49, 20.5, 1.40, 0.210, 1.19, 2.00, 8.50, 3.30, 2.0, 0.12],
                 [4.0, 3.49, 18.5, 1.20, 0.275, 1.03, 1.84, 10.25, 2.00, 2.0, 0.12],
```

```
                 [5.3, 4.84, 12.1, 1.90, 0.170, 1.87, 2.40, 5.95, 2.60, 14.8, 0.14],
                 [5.4, 4.84, 12.0, 1.64, 0.164, 1.68, 3.00, 4.30, 2.72, 14.5, 0.14],
                 [5.4, 4.84, 10.1, 2.30, 0.275, 2.08, 2.68, 5.45, 2.40, 0.9, 0.20],
                 [5.6, 4.48, 14.7, 2.35, 0.210, 2.55, 3.00, 3.75, 7.00, 2.0, 0.21],
                 [5.6, 4.48, 14.8, 2.35, 0.050, 1.32, 2.84, 5.10, 4.00, 0.4, 0.12],
                 [5.6, 4.48, 14.4, 2.50, 0.143, 2.38, 2.84, 4.05, 8.00, 3.8, 0.18],
                 [5.2, 3.48, 18.1, 1.50, 0.153, 1.20, 2.60, 9.00, 2.35, 14.5, 0.13],
                 [5.2, 3.48, 19.7, 1.65, 0.203, 1.73, 1.88, 5.30, 2.52, 12.5, 0.20],
                 [5.6, 3.48, 16.9, 1.40, 0.074, 1.15, 1.72, 9.85, 2.45, 8.0, 0.07],
                 [5.8, 2.63, 23.7, 1.65, 0.155, 1.58, 1.60, 3.60, 3.75, 4.9, 0.10],
                 [4.0, 2.63, 19.2, 0.90, 0.155, 0.96, 1.20, 4.05, 3.30, 0.2, 0.10],
                 [5.3, 2.63, 18.0, 1.60, 0.129, 1.68, 2.00, 4.40, 3.00, 3.6, 0.18],
                 [5.4, 4.46, 14.8, 2.45, 0.245, 2.15, 3.12, 7.15, 1.81, 12.0, 0.13],
                 [5.6, 4.46, 15.6, 1.65, 0.422, 1.42, 2.56, 7.25, 1.92, 5.2, 0.15],
                 [5.3, 2.80, 14.2, 1.65, 0.063, 1.62, 2.04, 5.30, 3.90, 10.2, 0.12],
                 [5.4, 2.80, 14.1, 1.25, 0.042, 1.62, 1.84, 3.10, 4.10, 8.5, 0.30],
                 [5.5, 2.80, 17.5, 1.05, 0.030, 1.56, 1.48, 2.40, 2.10, 9.6, 0.20],
                 [5.4, 2.57, 14.1, 2.70, 0.194, 2.77, 2.56, 4.25, 2.60, 6.9, 0.17],
                 [5.4, 2.57, 19.1, 1.60, 0.139, 1.59, 1.88, 5.80, 2.30, 4.7, 0.16],
                 [5.2, 2.57, 22.5, 0.85, 0.046, 1.65, 1.20, 1.55, 1.50, 3.5, 0.21],
                 [5.5, 1.26, 17.0, 0.70, 0.094, 0.97, 1.24, 4.55, 2.90, 1.9, 0.12],
                 [5.9, 1.26, 12.5, 0.80, 0.039, 0.80, 0.64, 2.65, 0.72, 0.7, 0.13],
                 [5.6, 2.52, 21.5, 1.80, 0.142, 1.77, 2.60, 4.50, 2.48, 8.3, 0.17],
                 [5.6, 2.52, 22.2, 1.05, 0.080, 1.17, 1.48, 4.85, 2.20, 9.3, 0.14],
                 [5.3, 2.52, 13.0, 2.20, 0.215, 1.85, 3.84, 8.75, 2.40, 13.0, 0.11],
                 [5.6, 3.24, 13.0, 3.55, 0.166, 3.18, 3.48, 5.20, 3.50, 18.3, 0.22],
                 [5.5, 3.24, 10.9, 3.30, 0.111, 2.79, 3.04, 4.75, 2.52, 10.5, 0.21],
                 [5.6, 3.24, 12.0, 3.65, 0.180, 2.40, 3.00, 5.85, 3.00, 14.5, 0.21],
                 [5.4, 1.56, 22.8, 0.55, 0.069, 1.00, 1.14, 2.85, 2.90, 3.3, 0.15],
                 [5.3, 1.56, 14.5, 2.05, 0.222, 1.49, 2.40, 4.55, 3.90, 6.3, 0.11],
                 [5.2, 1.56, 18.4, 1.05, 0.267, 1.17, 1.36, 4.60, 2.00, 4.9, 0.11],
                 [5.8, 4.12, 12.5, 5.90, 0.093, 3.80, 3.84, 2.90, 3.00, 22.5, 0.24],
                 [5.7, 4.12, 8.7, 4.25, 0.147, 3.62, 5.32, 3.00, 3.55, 19.5, 0.20],
                 [5.7, 4.12, 9.4, 3.85, 0.217, 3.36, 5.52, 3.40, 5.20, 1.3, 0.31],
                 [5.4, 2.14, 15.0, 2.45, 0.418, 2.38, 2.40, 5.40, 1.81, 20.0, 0.17],
                 [5.4, 2.14, 12.9, 1.70, 0.323, 1.74, 2.48, 4.45, 1.88, 1.0, 0.15],
                 [4.9, 2.03, 12.1, 1.80, 0.205, 2.00, 2.24, 4.30, 3.70, 5.0, 0.19],
                 [5.0, 2.03, 13.2, 3.65, 0.348, 1.95, 2.12, 5.00, 1.80, 3.0, 0.15],
                 [4.9, 2.03, 11.5, 2.25, 0.320, 2.25, 3.12, 3.40, 2.50, 5.1, 0.18]]

        data = np.asmatrix(data)
        get_results(data)
```

```
- Using S:
Number of Components: 11
Percentage Contributions:
   1. 62.2544 %
   2. 23.6949 %
```

```
   3. 6.4974 %
   4. 4.5549 %
   5. 1.4624 %
   6. 0.9402 %
   7. 0.3025 %
   8. 0.2079 %
   9. 0.0732 %
   10. 0.0106 %
   11. 0.0017 %
Number of components to be retained = 2

- Using R:
Number of Components: 11
Percentage Contributions:
   1. 39.7357 %
   2. 15.7454 %
   3. 12.4738 %
   4. 9.6415 %
   5. 5.9236 %
   6. 4.4609 %
   7. 4.1889 %
   8. 3.2388 %
   9. 2.5955 %
   10. 1.5199 %
   11. 0.4758 %
Number of components to be retained = 4
```

```
In [6]: # ^_^ Thank You
```