Principal Component Analysis

import library

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib import cm
```

load data

In [ ]:

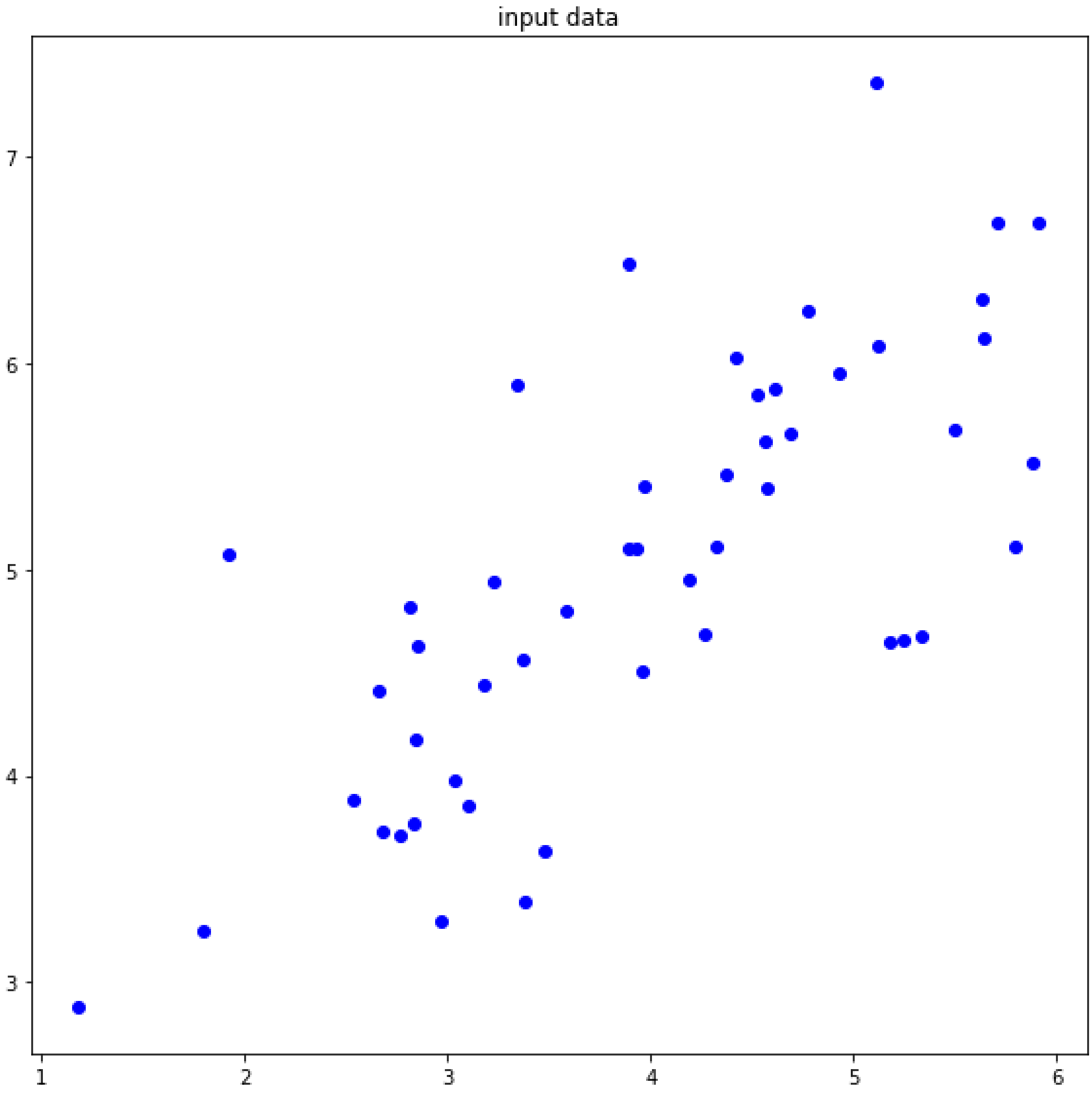
```
fname_data  = 'assignment_12_data.txt'
feature0    = np.genfromtxt(fname_data, delimiter=',')

number_data     = np.size(feature0, 0)
number_feature  = np.size(feature0, 1)

print('number of data : {}'.format(number_data))
print('number of feature : {}'.format(number_feature))
```

```
number of data : 50
number of feature : 2
```

plot the input data

In [ ]:

```
plt.figure(figsize=(8,8))
plt.title('input data')

x0 = feature0[:,0]
y0 = feature0[:,1]

plt.scatter(x0, y0, color='blue')

plt.tight_layout()
plt.show()
```



Normalization (Z-scoring)

- shape of feature = $n \times m$ where n is the number of data and m is the dimension of features

In [ ]:

```
def normalize(feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    feature_normalize = np.zeros((number_data, number_feature), dtype = np.float64)
    feature_normalize[:, 0] = (x0 - np.mean(x0)) / np.std(x0)
    feature_normalize[:, 1] = (y0 - np.mean(y0)) / np.std(y0)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return feature_normalize
```

In [ ]:

```
feature = normalize(feature0)

x   = feature[:, 0]
y   = feature[:, 1]

min_x = np.min(x)
min_y = np.min(y)

max_x = np.max(x)
max_y = np.max(y)
```

compute covariance matrix

- shape of feature = $n \times m$ where n is the number of data and m is the dimension of features

In [ ]:

```
def compute_covariance(feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    Sigma = np.matmul(feature.T, feature) / np.shape(feature)[0]

    #Sigma = np.cov(feature.T)
    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return Sigma
```

compute principal components

- `np.linalg.eig`
- `argsort()`
- return the eigenvalues and the eigenvectors in a decreasing order according to the eigenvalues

In [ ]:

```
def compute_principal_component(feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    eigenvalues, eigenvectors = np.linalg.eig(compute_covariance(feature))

    principal_component_1 = eigenvalues[eigenvalues.argsort()[::-1]][0] * eigenvectors[:,eigenvalues.argsort()[::-1]][:,0]
    principal_component_2 = eigenvalues[eigenvalues.argsort()[::-1]][1] * eigenvectors[:,eigenvalues.argsort()[::-1]][:,1]

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return (principal_component_1, principal_component_2)
```

compute the projection of point onto the axis

- `np.matmul`
- `np.dot`
- shape of feature = $n \times m$ where n is the number of data and m is the dimension of features
- shape of vector = $m \times 1$ where m is the dimension of features

In [ ]:

```
def compute_projection_onto_line(feature, vector):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    n = np.shape(feature)[0]
    m = np.shape(feature)[1]
    projection = np.zeros((n,m))

    for i in range(n):
        projection[i, :] = (np.dot(feature[i, :], vector) / np.sum(vector**2)) * vector

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return projection
```

compute the principal components and the projection of feature

In [ ]:

```
(principal_component_1, principal_component_2) = compute_principal_component(feature)

projection1 = compute_projection_onto_line(feature, principal_component_1)
projection2 = compute_projection_onto_line(feature, principal_component_2)
```

functions for presenting the results

In [ ]:

```
def function_result_01():

    plt.figure(figsize=(8,8))
    plt.title('data normalized by z-scoring')
    plt.scatter(x, y, color='blue')

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

```
def function_result_02():

    plt.figure(figsize=(8,8))
    plt.title('principal components')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    plt.scatter(x, y, color='blue')
    plt.quiver(0, 0, principal_component_1[0], principal_component_1[1], color='r', scale = 5, alpha = 0.8)
    plt.quiver(0, 0, principal_component_2[0], principal_component_2[1], color='g', scale = 5, alpha = 0.8)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

```
def function_result_03():

    plt.figure(figsize=(8,8))
    plt.title('first principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    plt.scatter(x, y, color='blue')
    plt.axline((0,0), principal_component_1, color = 'r')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

```
def function_result_04():

    plt.figure(figsize=(8,8))
    plt.title('second principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    plt.scatter(x, y, color='blue')
    plt.axline((0,0), principal_component_2, color = 'r')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

```
def function_result_05():

    plt.figure(figsize=(8,8))
    plt.title('projection onto the first principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    plt.scatter(x, y, color='blue')
    plt.axline((0,0), principal_component_1, color = 'r')
    plt.plot(projection1[:, 0], projection1[:, 1], 'o', color = 'g')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

```
def function_result_06():

    plt.figure(figsize=(8,8))
    plt.title('projection onto the second principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    plt.scatter(x, y, color='blue')
    plt.axline((0,0), principal_component_2, color = 'r')
    plt.plot(projection2[:, 0], projection2[:, 1], 'o', color = 'g')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

```
def function_result_07():

    plt.figure(figsize=(8,8))
    plt.title('projection onto the first principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    plt.scatter(x, y, color='blue')
    plt.axline((0,0), principal_component_1, color = 'r')
    plt.plot(projection1[:, 0], projection1[:, 1], 'o', color = 'g')
    for i in range(np.shape(feature)[0]):
        plt.plot((x[i], projection1[i, 0]), (y[i], projection1[i, 1]), color = 'gray')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

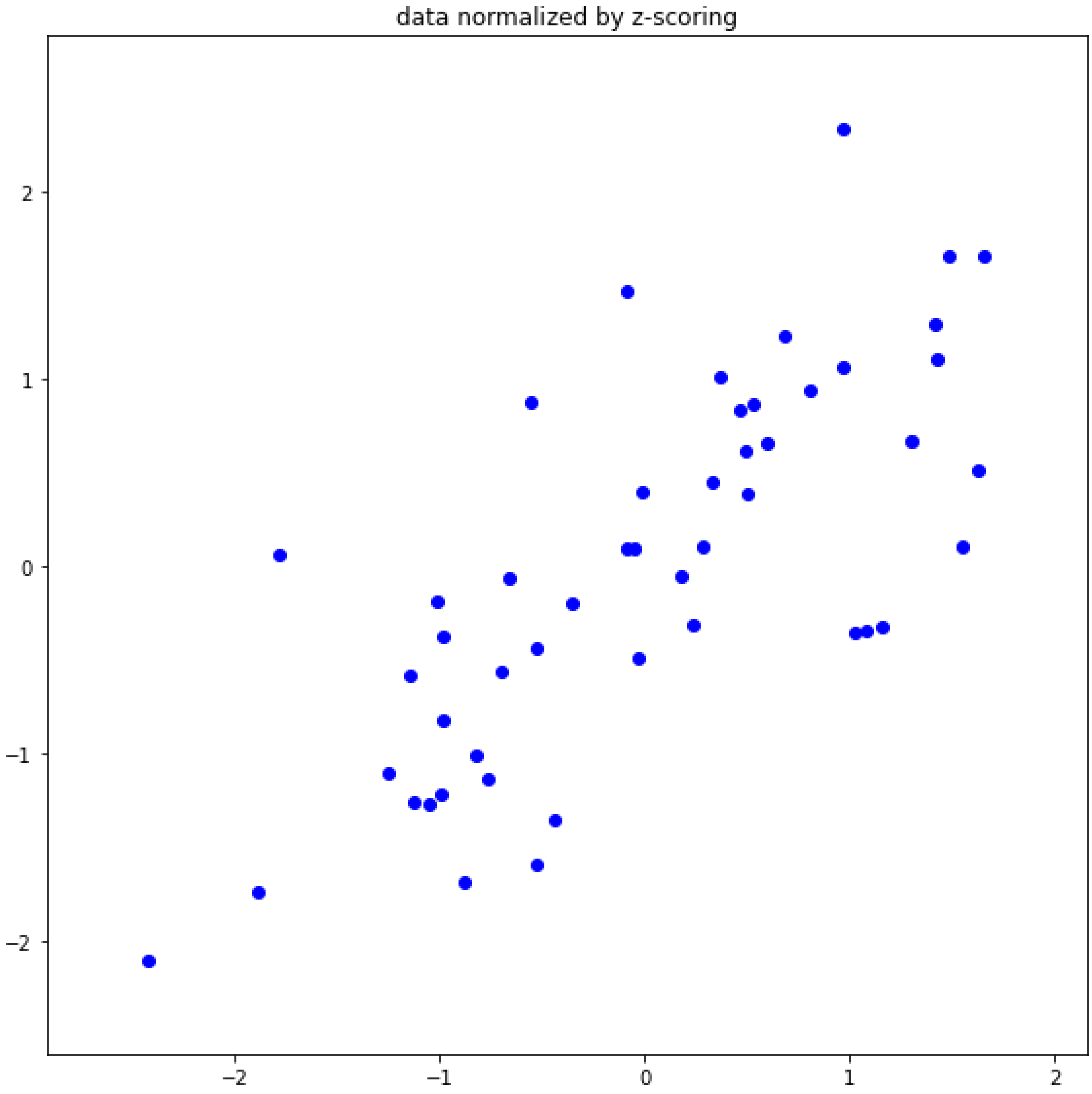
```
def function_result_08():

    plt.figure(figsize=(8,8))
    plt.title('projection to the second principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    plt.scatter(x, y, color='blue')
    plt.axline((0,0), principal_component_2, color = 'r')
    plt.plot(projection2[:, 0], projection2[:, 1], 'o', color = 'g')
    for i in range(np.shape(feature)[0]):
        plt.plot((x[i], projection2[i, 0]), (y[i], projection2[i, 1]), color = 'gray')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

results

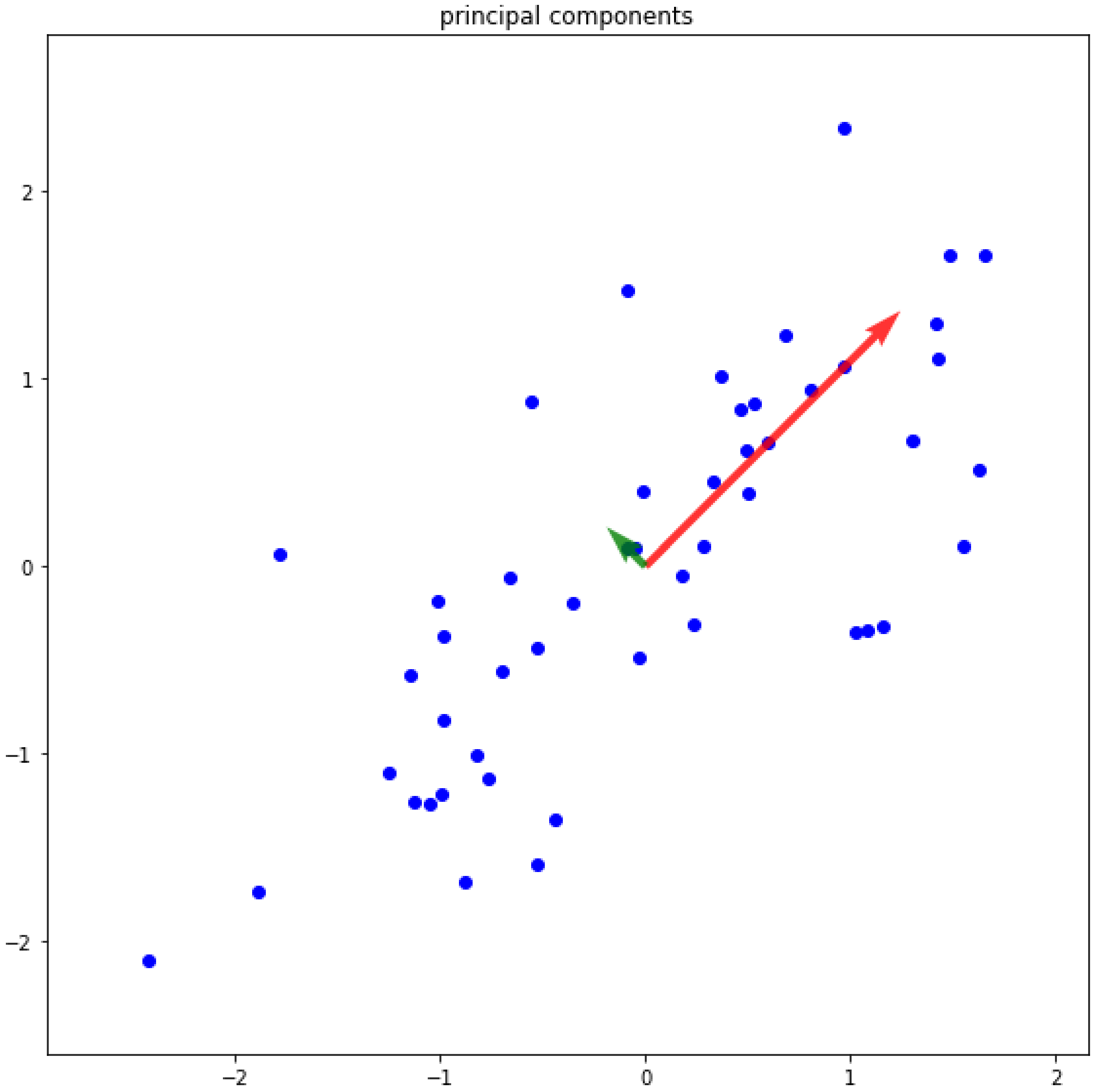
In [ ]:

```
number_result = 8

for i in range(number_result):
    title = '## [RESULT {:02d}]'.format(i+1)
    name_function = 'function_result_{:02d}()'.format(i+1)

    print('**************************************************')
    print(title)
    print('**************************************************')
    eval(name_function)
```

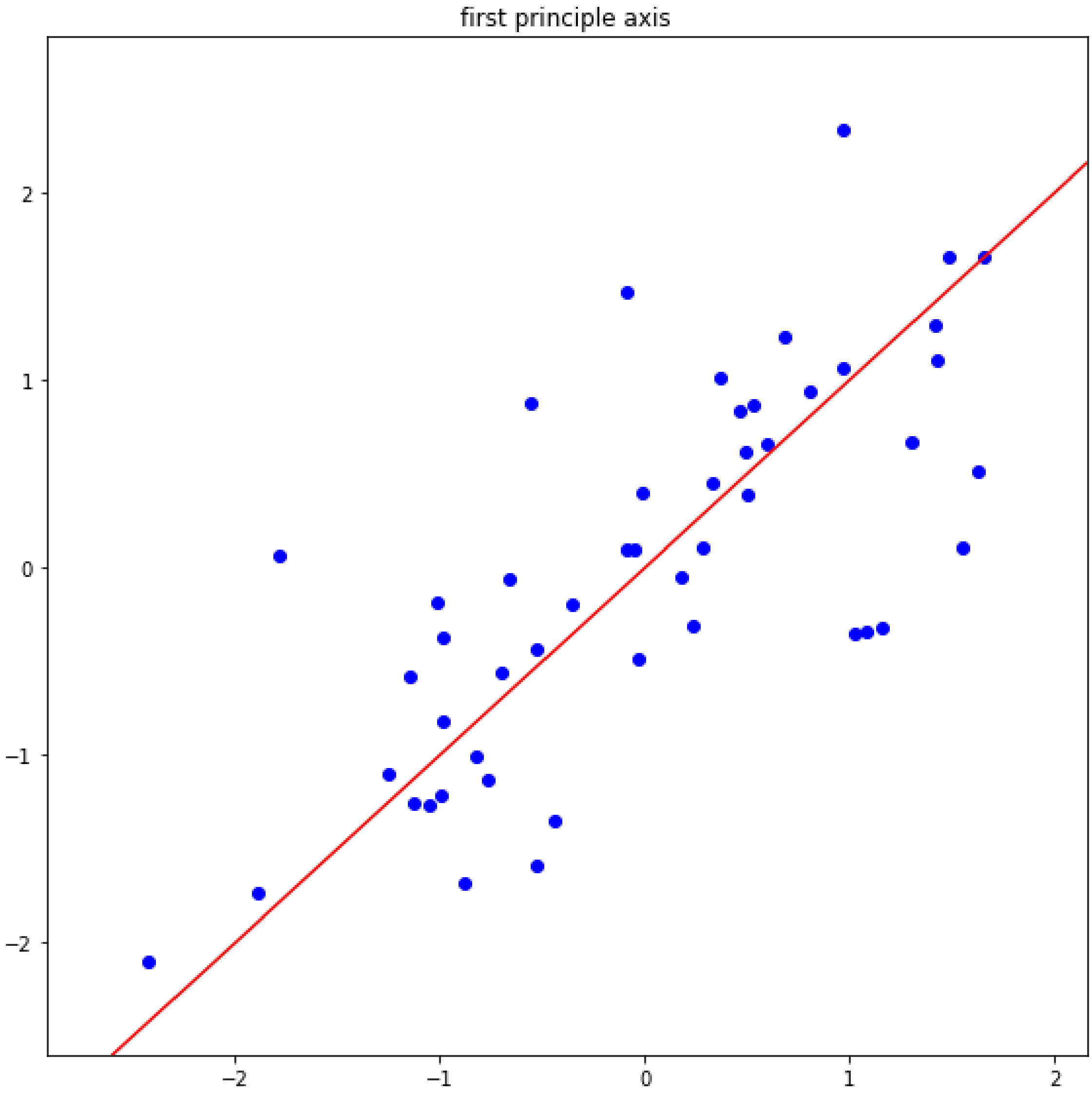
```
**************************************************
## [RESULT 01]
**************************************************
```



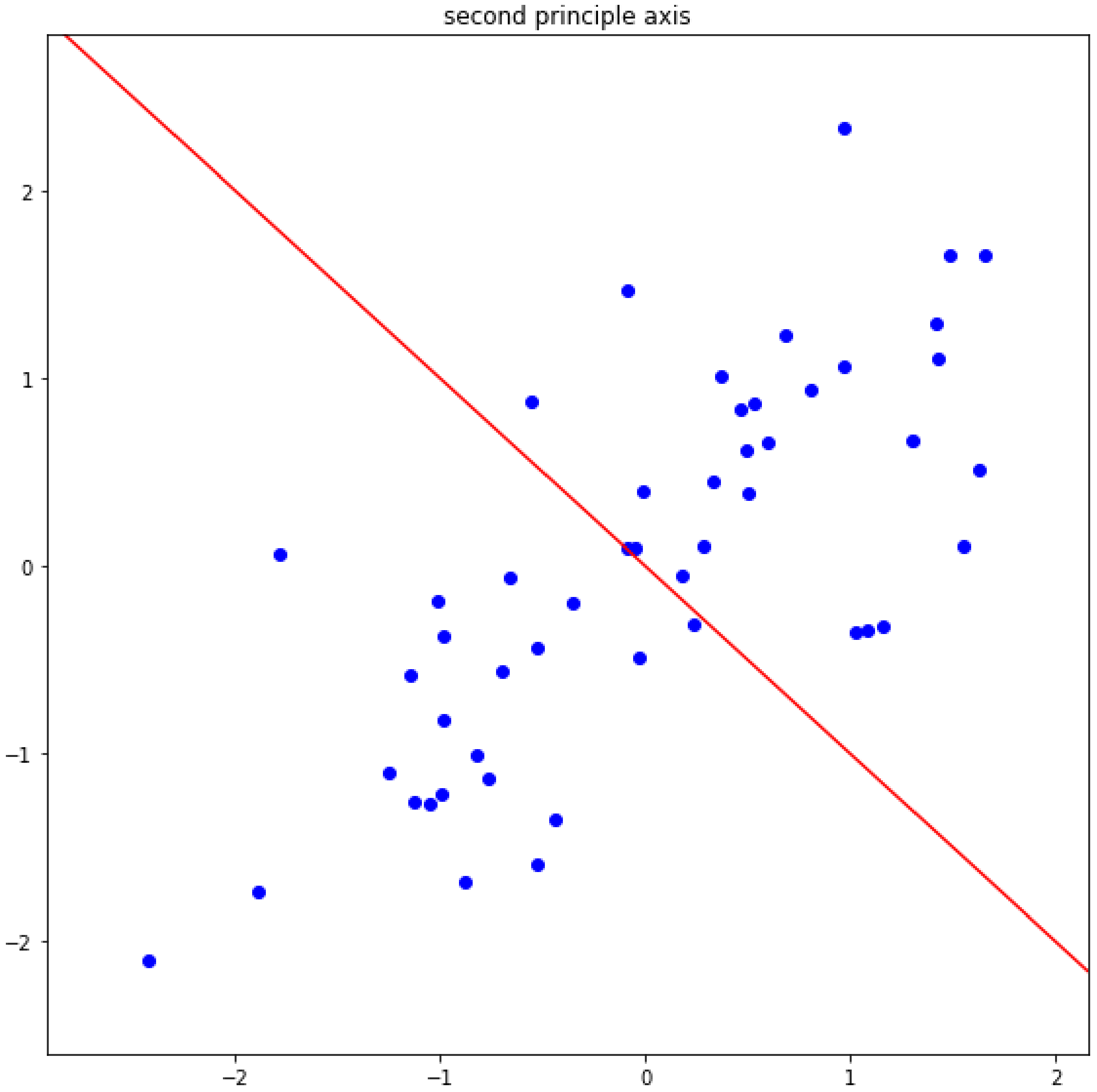
```
**************************************************
## [RESULT 02]
**************************************************
```



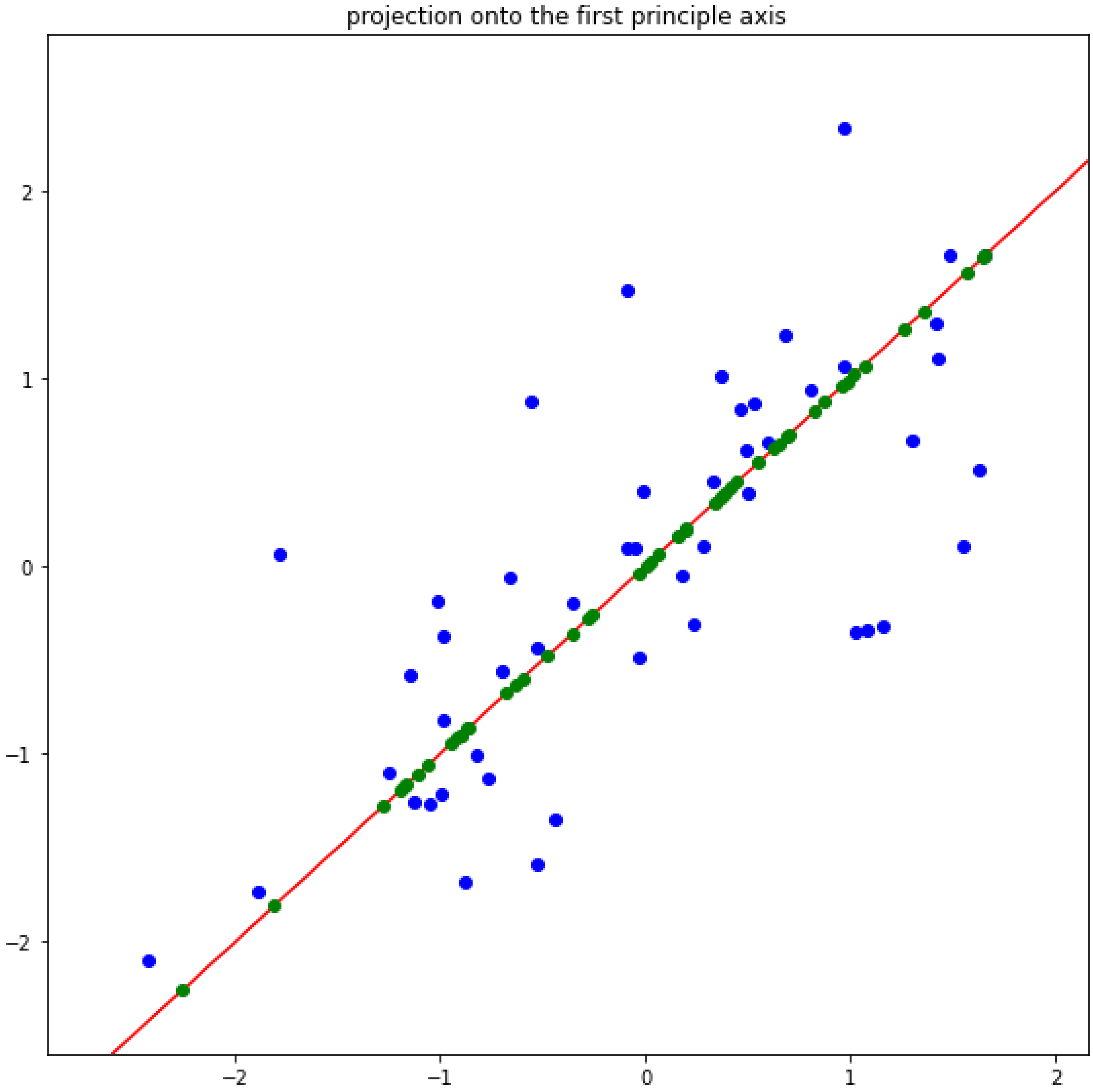
```
*************************************************
## [RESULT 03]
*************************************************
```



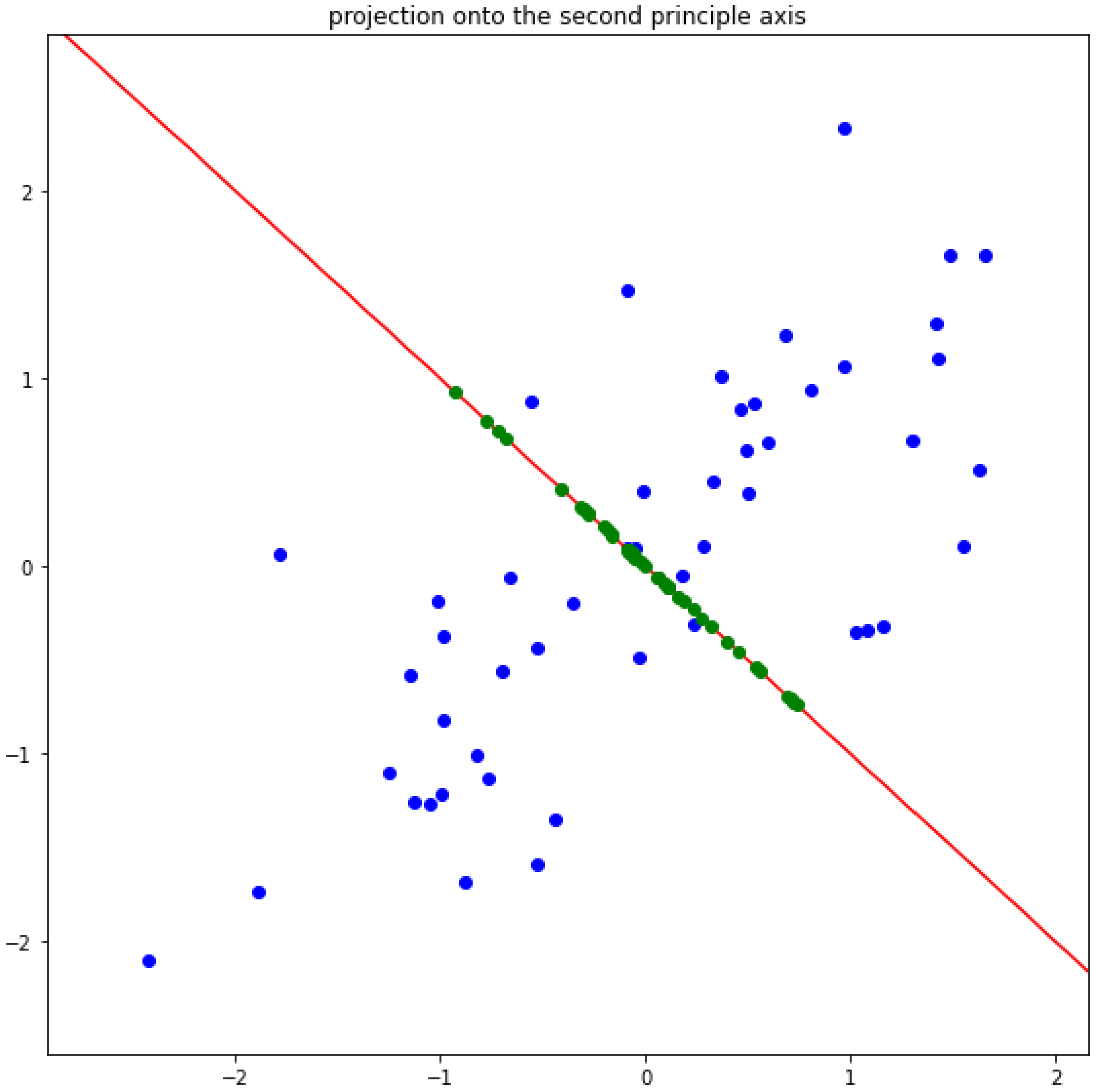
```
**************************************************
## [RESULT 04]
**************************************************
```



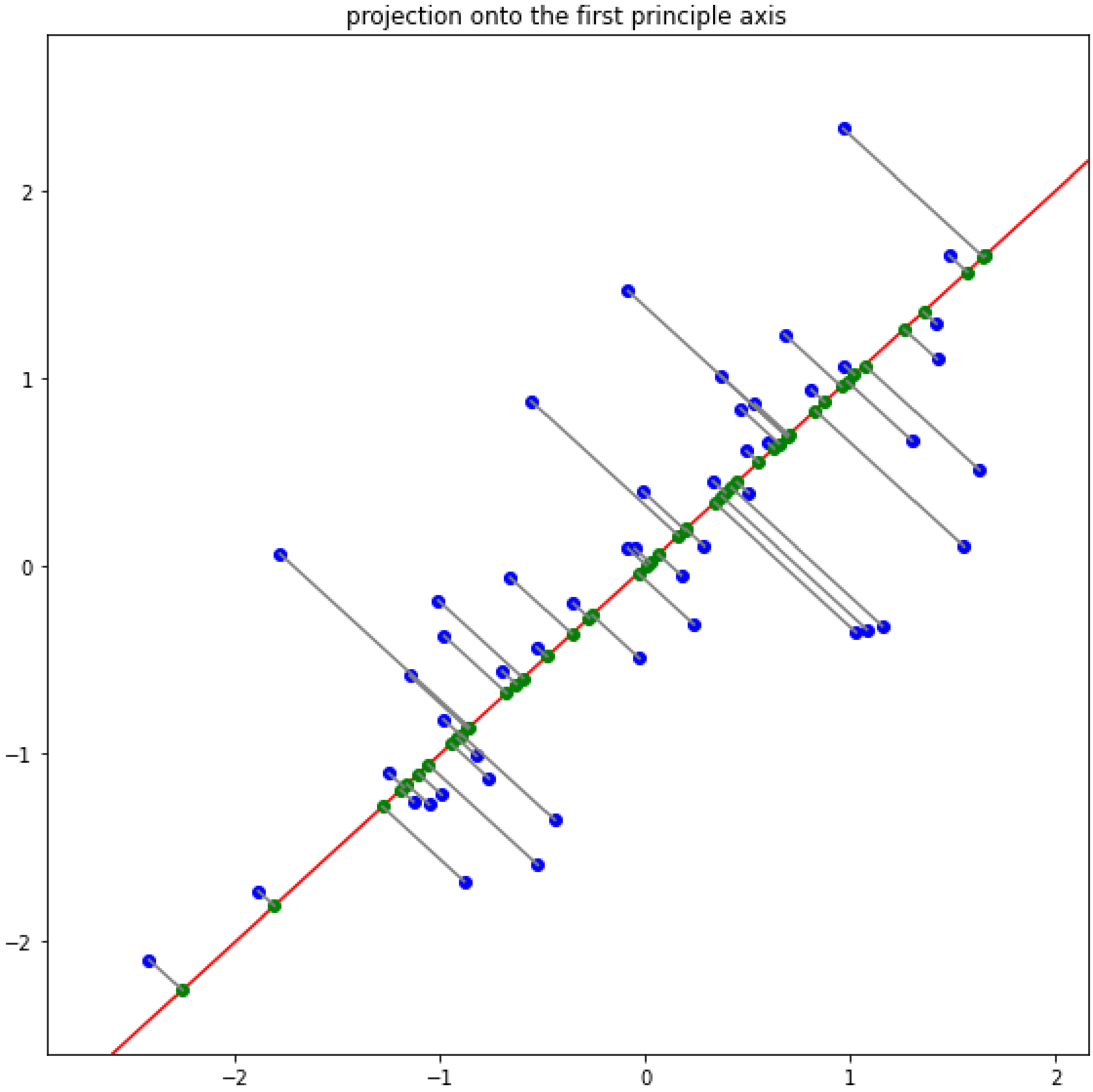
```
**************************************************
## [RESULT 05]
**************************************************
```



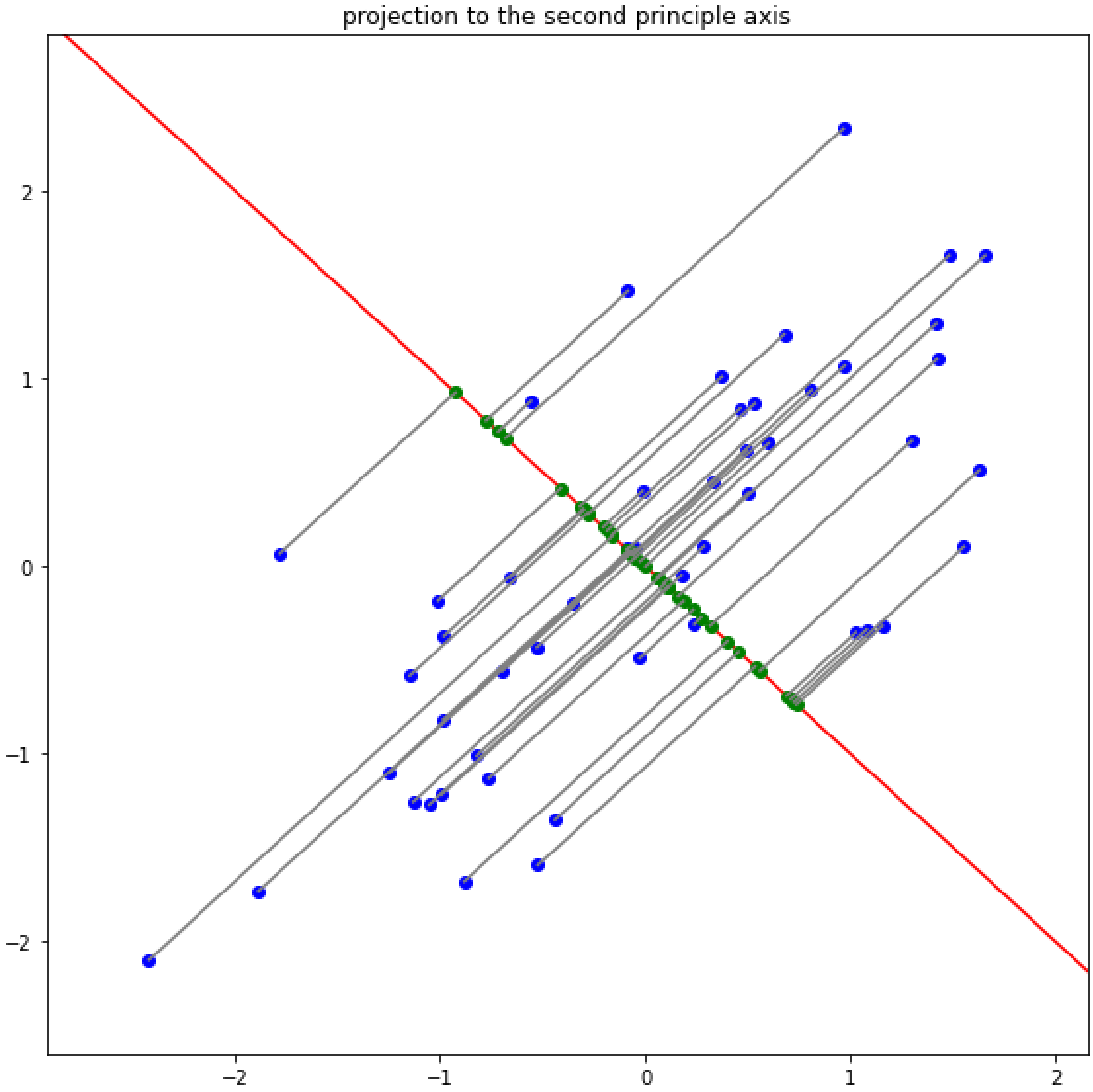
```
**************************************************
## [RESULT 06]
**************************************************
```



```
*****************************************************
## [RESULT 07]
*****************************************************
```



```
**************************************************
## [RESULT 08]
**************************************************
```


projection to the second principle axis
2
1
0
−1
−2
−2
−1
0
1
2

In [ ]: