1. [1 pt] Load the data from the npz file. The file contains a data in a dictionary format. Use a key data to retrieve a matrix. The type of the matrix should be the NumPy array and the shape should be (300, 2) (i.e., 300 data instances).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

file_path = 'pca_data.npz'
data_dict = np.load(file_path)

data_matrix = data_dict['data']
print("Shape of the data matrix:", data_matrix.shape)
```

```
Shape of the data matrix: (300, 2)
```

2. [1pt] Use PCA from scikit-learn to compute two principal components. Report the computed principal components. There could be two different answers for this. One from scaled data and another from the raw data (non-scaled). Either is okay.

```
file_path = 'pca_data.npz'
data_dict = np.load(file_path)
data_matrix = data_dict['data']

pca = PCA(n_components=2)
principal_components = pca.fit_transform(data_matrix)

print("First Principal Component:\n", pca.components_[0])
print("Second Principal Component:\n", pca.components_[1])
```

```
First Principal Component:
 [0.83774478 0.54606197]
Second Principal Component:
 [-0.54606197  0.83774478]
```

3. [2pt] Implement your own version of PCA algorithm without using scikit-learn API.

```
file_path = 'pca_data.npz'
data_dict = np.load(file_path)
data_matrix = data_dict['data']

data_mean = np.mean(data_matrix, axis=0)
data_centered = data_matrix - data_mean
covariance_matrix = np.cov(data_centered, rowvar=False)
eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
sorted_indices = np.argsort(eigenvalues)[::-1]
principal_components = eigenvectors[:, sorted_indices[:2]]
transformed_data = np.dot(data_centered, principal_components)

print("First Principal Component:\n", principal_components[:, 0])
print("Second Principal Component:\n", principal_components[:, 1])
```

```
First Principal Component:
 [-0.83774478 -0.54606197]
Second Principal Component:
 [ 0.54606197 -0.83774478]
```

4. [1pt] Produce a plot with data instances and computed principal components. • for plotting data instances, use either plot or scatter of Matplotlib. • for plotting the principal components, use quiver. Plot two arrows that are centered around mean, in the direction of principal components, and of lengths proportional to explained variance. An example is shown below. To obtain the information on the mean, the principal component directions, and the explained variances, you will have to read the scikit-learn document carefully.

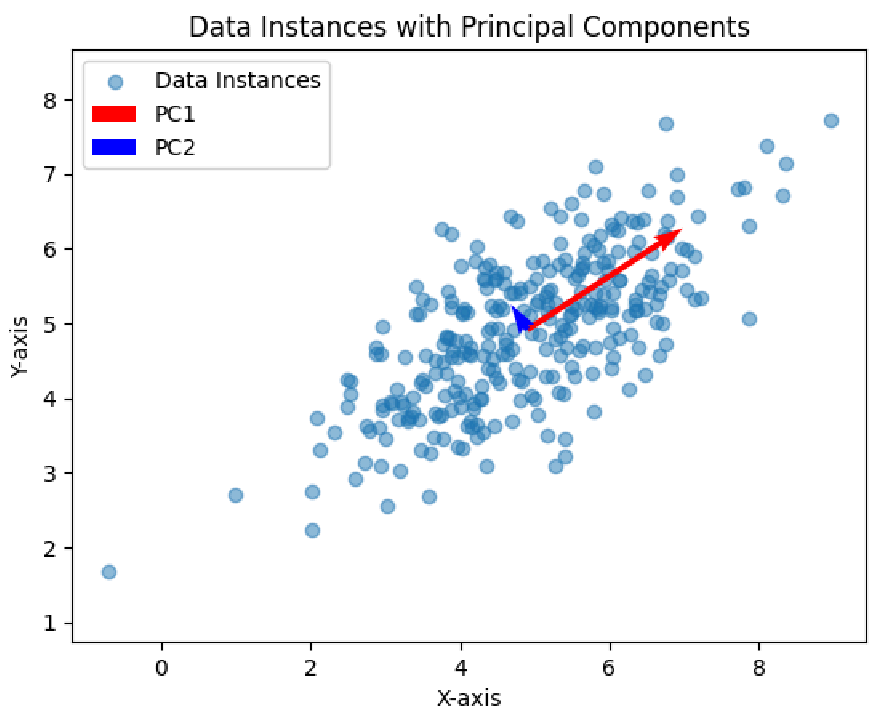
```
file_path = 'pca_data.npz'
data_dict = np.load(file_path)
data_matrix = data_dict['data']

pca = PCA(n_components=2)
pca.fit(data_matrix)
principal_components = pca.components_
explained_variance = pca.explained_variance_
data_mean = np.mean(data_matrix, axis=0)

plt.scatter(data_matrix[:, 0], data_matrix[:, 1], alpha=0.5, label="Data Instances")

plt.quiver(
    data_mean[0], data_mean[1],
    principal_components[0, 0], principal_components[0, 1],
    scale=1/explained_variance[0], scale_units='xy', angles='xy', color='r', label='PC1'
)
plt.quiver(
    data_mean[0], data_mean[1],
    principal_components[1, 0], principal_components[1, 1],
    scale=1/explained_variance[1], scale_units='xy', angles='xy', color='b', label='PC2'
)

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.title("Data Instances with Principal Components")
plt.axis('equal')
plt.show()
```



Data Instances with Principal Components
Data Instances
PC1
PC2
X-axis
Y-axis
0
2
4
6
8
1
2
3
4
5
6
7
8