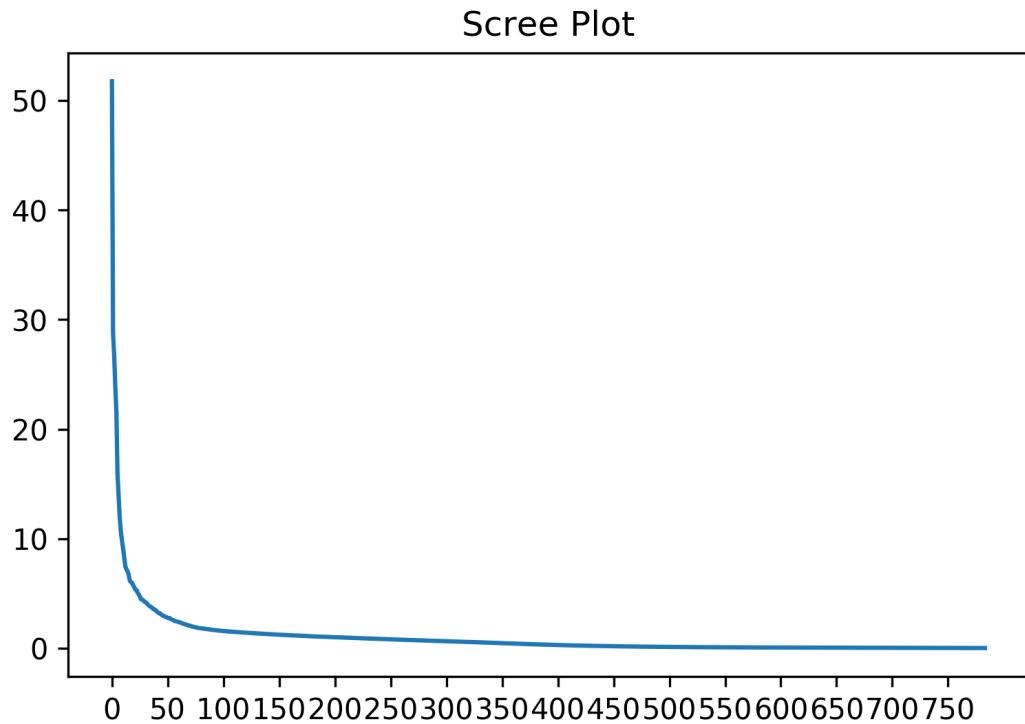
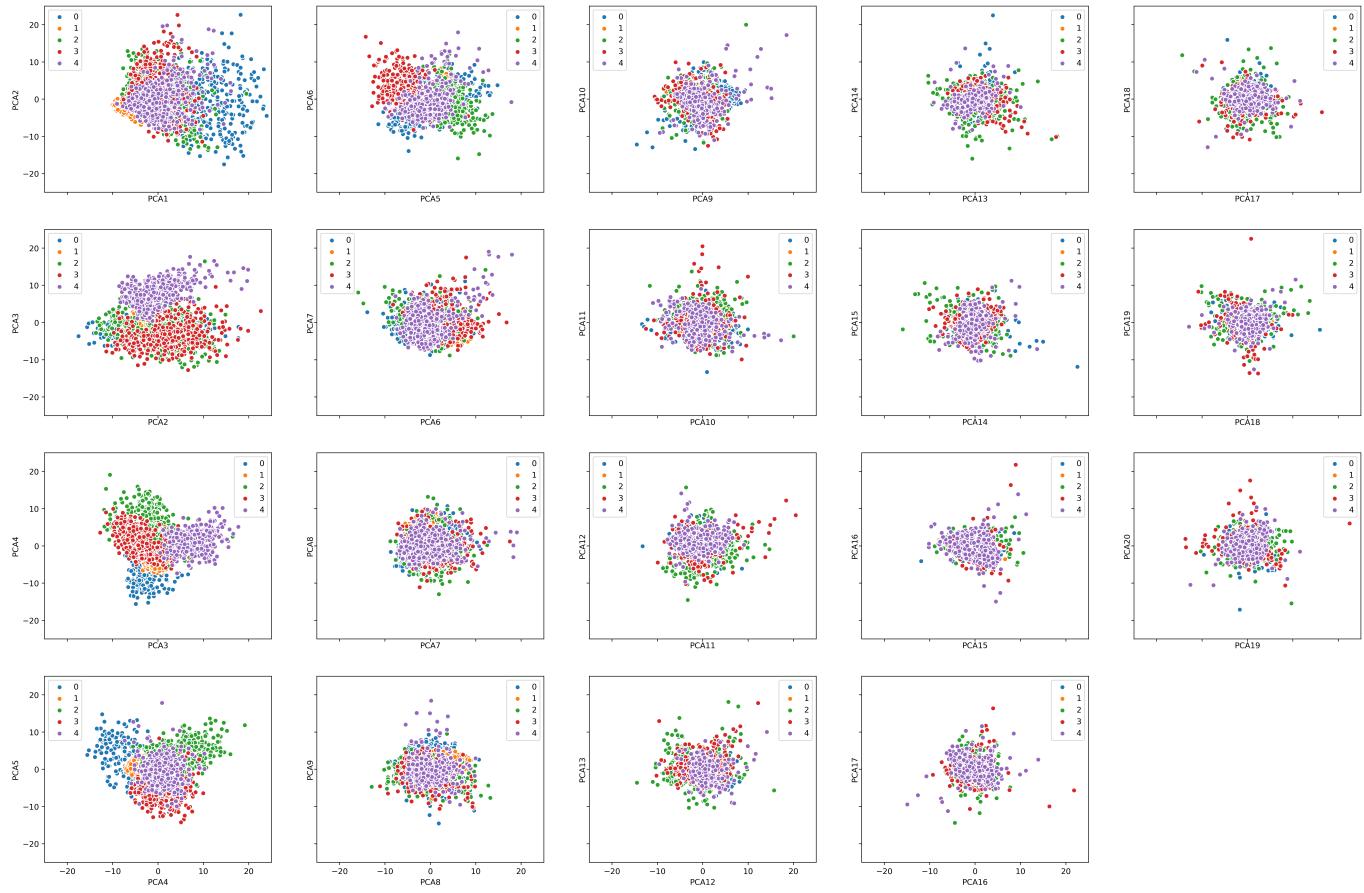


Question 2:**Scree Plot**

According to the scree plot, the explained variance drops sharply from 0 to 50. A cut-off around 10 is good since after that, the amount of variance explained by the selected component is relatively small.

Projected Data With Top 20 Eigenvalues in PCA

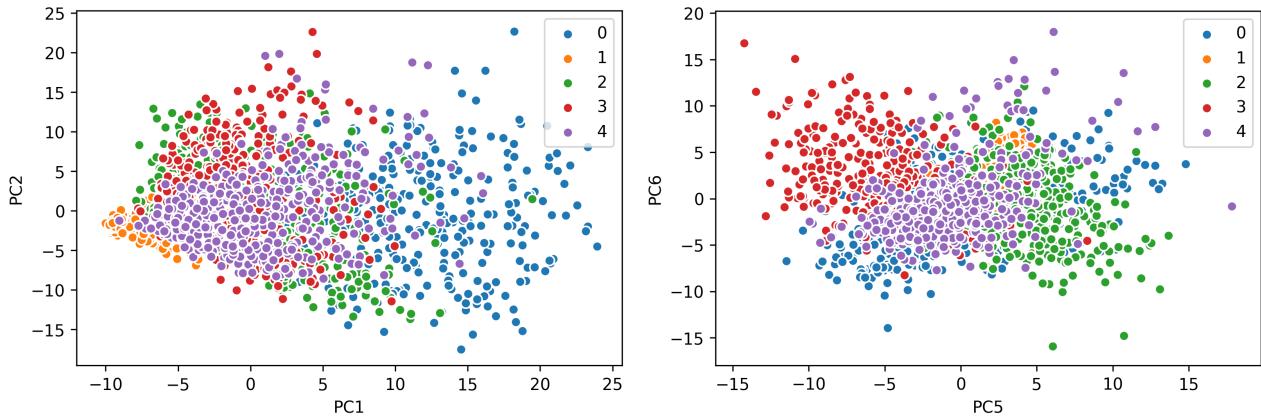


We can see some clear separations between classes in the first 6 plots(PCA1 vs PCA2 to PC5 vs PC6).

But after that almost all points are concentrated in the middle and we cannot tell them apart.

Hence, keeping the first 6 components is a good choice. It is clearer to find the cut-off using pair plot since the slope is too large in the scree plot, making it difficult to find a specific cut-off value.

PC1,2 Compared to PC5,6



The first plot(PC1 vs PC2) mainly separates number 0 from other classes. It is difficult to distinguish the other 4 classes from the first plot as they all cluster on the left. Actually, PC2 does not help much to classify number 0, since the border between number 0 and others are almost vertical on the graph($PC1 = 10$).

The second plot(PC5 vs PC6) makes some contribution to separating number 2 and number 3 from other classes. But it is not sufficient to use only PC5 and PC6 since there is large overlap in the middle between all numbers.

Although the first plot seems more orderless than the second plot, we can use it along to separate one class. But we cannot separate classes using only the second plot.

SVD, PCA and Dual PCA Implementation

```
# Singular value decomposition
# input: an n*t array , n is the dimensionality and t is
# the number of training examples
def svd_np(x):
    eigValueU, eigVectorU = np.linalg.eigh(np.dot(x, x.T))
    eigValueV, eigVectorV = np.linalg.eigh(np.dot(x.T, x))
    idxU = eigValueU.argsort()[:-1]
    idxV = eigValueV.argsort()[:-1]
    return eigVectorU[:, idxU],
           np.sqrt(eigValueV[idxV] if x.shape[0] > x.shape[1] else eigValueU[idxU]),
           eigVectorV[:, idxV].T
```

```
# First get U, sigma and Vt from SVD
# s is a t*n array
U, sigma, Vt = svd_np(s.T)

# PCA
s . dot (U)

# Dual PCA
np . diag (sigma) . dot (Vt [: s . shape [1] , :]) . T
```

Assume that we have already known the value of U , Σ and V^T , run PCA and dual PCA 1000 times each and get the average runtime.

PCA: 22359.35 microseconds

Dual PCA: 22124.32 microseconds

Under this specific dataset, their performance is basically the same.

Prove that PCA is the best linear method for reconstruction

Optimization problem: maximize $\text{tr}(U^T S U)$ s.t. $U^T U = I$, where $S = \bar{X} \bar{X}^T$ and \bar{X} is the normalized input data.

In other words, maximize the variance of the projection under constraint $U^T U = I$.

$$\begin{aligned} \mathcal{L}(U, \lambda) &= \text{tr}(U^T S U) - \text{tr}(\lambda(U^T U - I)) \\ \frac{\partial \mathcal{L}}{\partial U} &= 2S U - 2\lambda U \end{aligned} \tag{1}$$

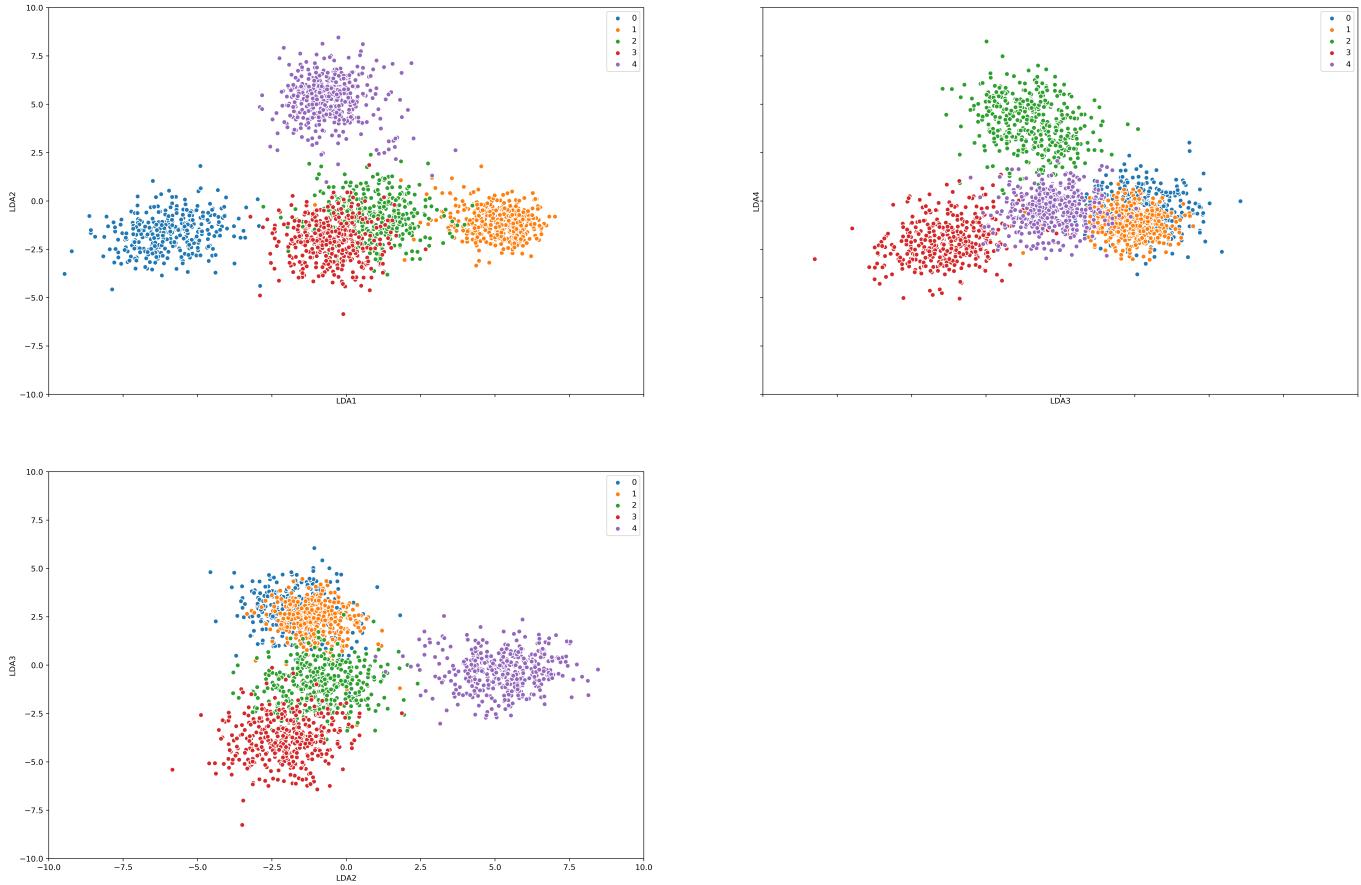
$$\text{let } \frac{\partial \mathcal{L}}{\partial U} = 0$$

$$S U = \lambda U$$

The result indicates that to maximize the equation, U should take the value of the eigenvector of

S with the largest eigenvalue.

Projected Data With Top 4 Eigenvalues in LDA



LDA1 separates number 0 and number 1. LDA2 separates number 4. LDA2 separates number 3 and LDA4 separates number 2.

Each LDA direction is mainly responsible for separating one or two classes, but that is not the case with each principle component.

Compare LDA Optimization With PCA Optimization