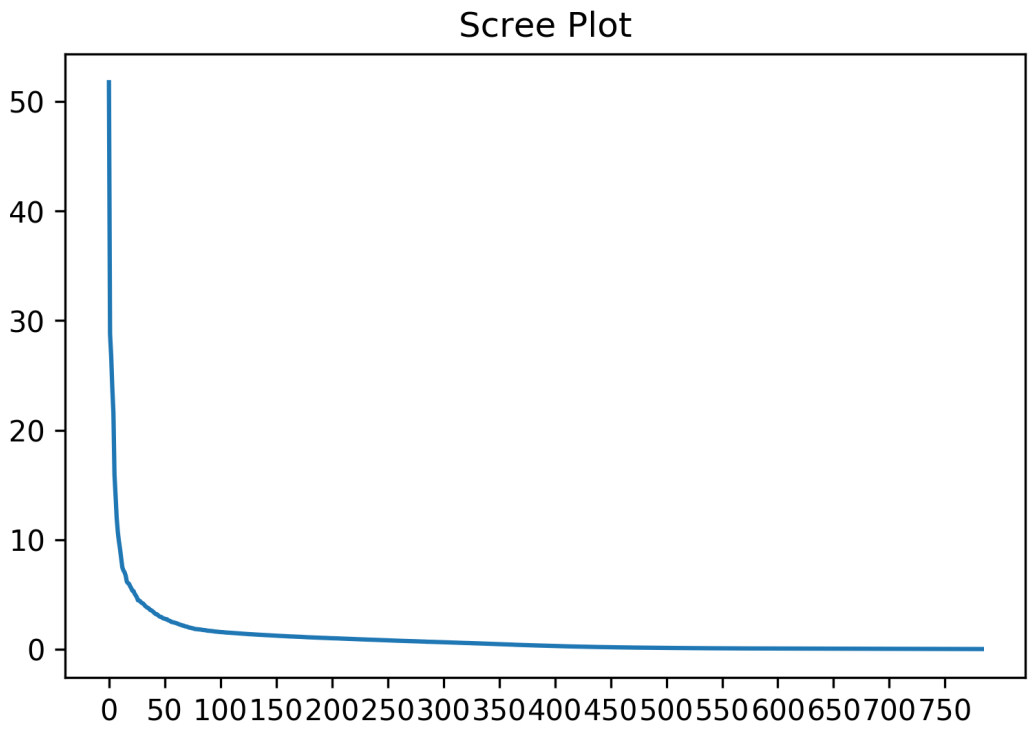
# The source code is at the end of this document

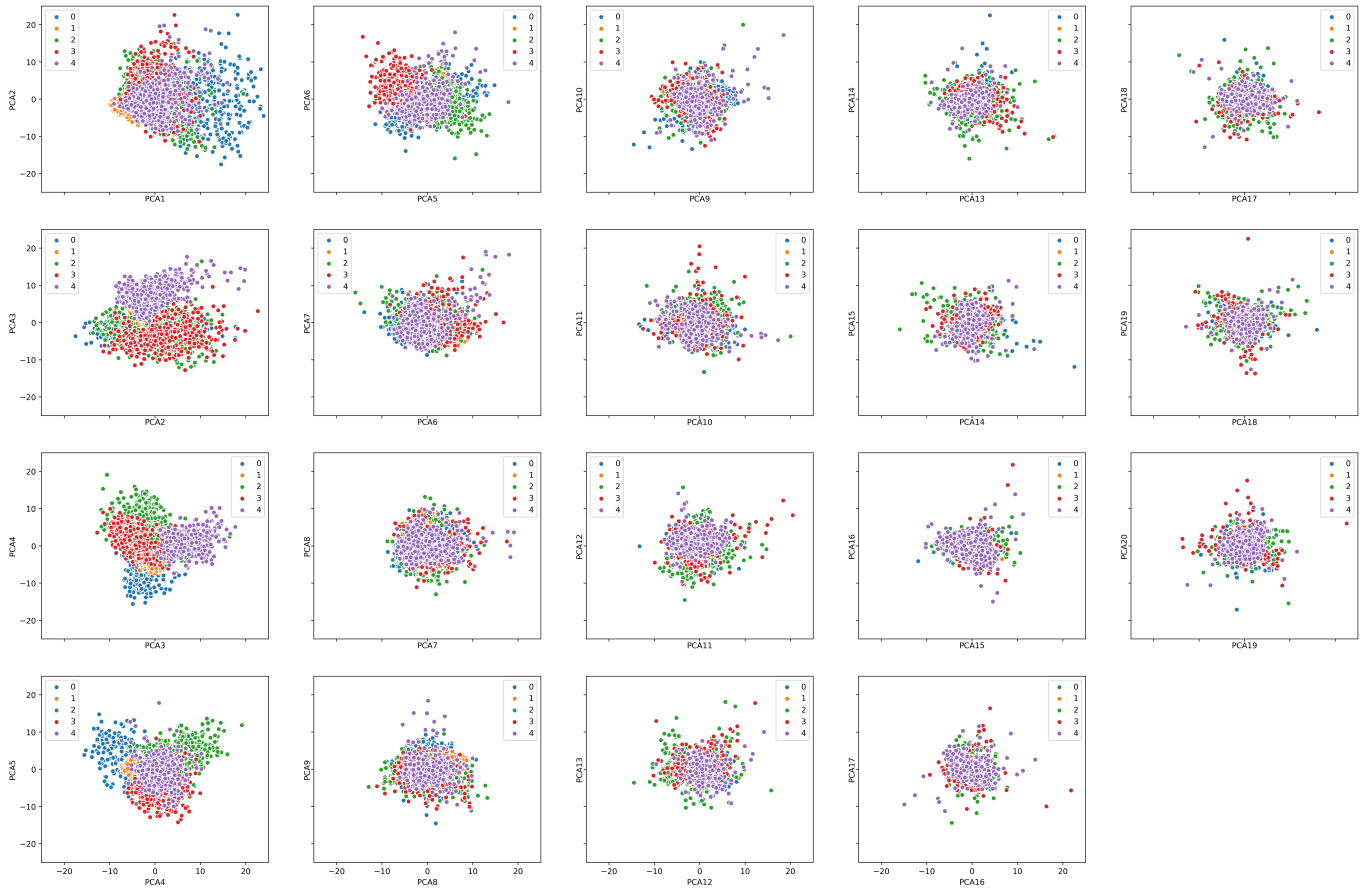
## Question 2:

### Scree Plot



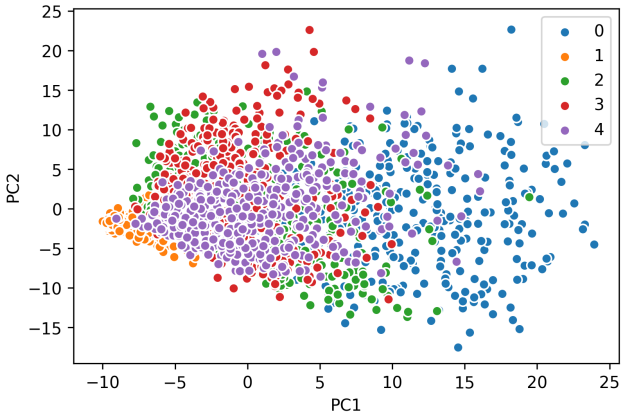

According to the scree plot, the explained variance drops sharply from 0 to 50. A cut-off around 10 is good since after that, the amount of variance explained by the selected component is relatively small.

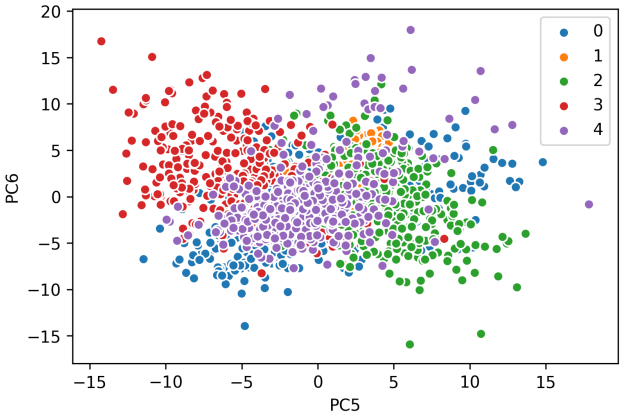
### Projected Data With Top 20 Eigenvalues in PCA




We can see some clear separations between classes in the first 6 plots(PC1 vs PC2 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

```
1  # Singular value decomposition
2  # input: an n*t array, n is the dimensionality and t is
3  # the number of training examples
4  def svd_np(x):
5      eigValueU, eigVectorU = np.linalg.eigh(np.dot(x, x.T))
6      eigValueV, eigVectorV = np.linalg.eigh(np.dot(x.T, x))
7      idxU = eigValueU.argsort()[::-1]
8      idxV = eigValueV.argsort()[::-1]
9      return eigVectorU[:,idxU],
10     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

# check if (sigma * Vt).T has the same result with build in PCA
print(np.allclose(np.absolute(np.diag(sigma).dot(Vt[:s.shape[1],:]).T), np.absolute(PCA().fit_transform(s))))
# True

# check if (U.T.dot(s.T)).T, which is s.dot(U), has the same result with build in PCA
print(np.allclose(np.absolute(s.dot(U)), np.absolute(PCA().fit_transform(s))))
# True
```

Assume that we have already known the value of $U$, $\Sigma$ 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. In theory, dual PCA is only faster than PCA when the number of features >> number of samples because maxtrix multiplication is simpler for Dual PCA under this condition.

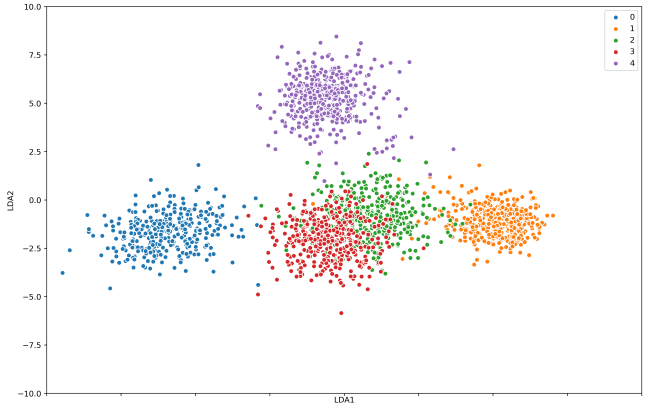
**Prove that PCA is the best linear method for reconstruction**

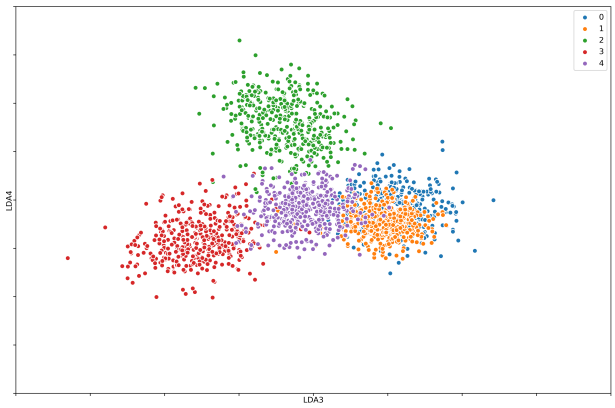
Optimization problem: minimize the reconstruction error. $\underset{\mathbf{U}}{\text{Minimize}} \left\| \overline{X} - UU^T\overline{X} \right\|_F^2$ s.t. $U^TU = I$.

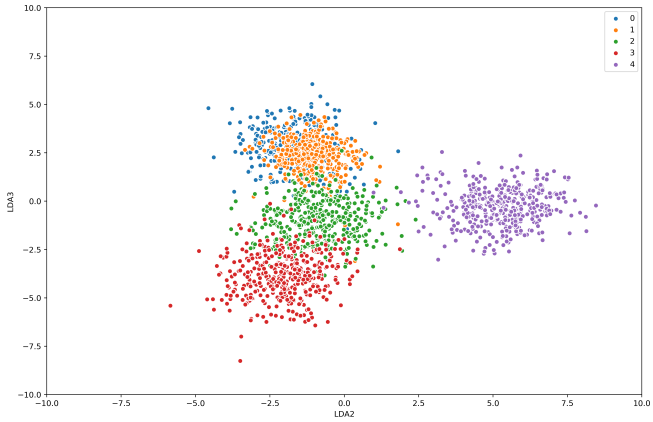
$$\begin{aligned}
\left\| \overline{X} - UU^T\overline{X} \right\|_F^2 &= tr((\overline{X} - UU^T\overline{X})(\overline{X} - UU^T\overline{X})^T) \\
&= tr((\overline{X} - UU^T\overline{X})(\overline{X}^T - \overline{X}^TUU^T)) \\
&= tr(\overline{X}\overline{X}^T) - 2tr(\overline{X}UU^T\overline{X}^T) + tr(\overline{X}UU^TUU^T\overline{X}^T) \\
&= tr(\overline{X}\overline{X}^T) - tr(\overline{X}UU^T\overline{X}^T) \\
&= tr(\overline{X}\overline{X}^T) - tr(U^T\overline{X}^T\overline{X}U) \\
&= const - tr(U^TSU)
\end{aligned} \tag{1}$$

The result indicates that minimizing the reconstruction error is equivalent to maximizing the trace of $U^TSU$, which is exactly what PCA is doing. Therefore, PCA is the best linear method for reconstruction.

**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**

FDA optimization: $\underset{\mathbf{U}}{\text{Maximize}}\ tr(U^T S U)$ s.t. $U^T S_W U = I$

PCA optimization: $\underset{\mathbf{U}}{\text{Maximize}}\ tr(U^T S U)$ s.t. $U^T U = I$

FDA and PCA can be treated as the same optimization problem with different constraints. Since $S_W$ needs extra information(the label of the data), FDA is supervised linear transformation while

PCA is unsupervised.

The ability of a component to separate different classes apart decreases as the number increases for both PCA and LDA becauese we sort the eigenvectors based on the magnitute of eigenvalues. Since the solution of FDA is the eigenvalues of $S_W^{-1}S_B$ and $S_B$ has rank $k-1$, where k is the number of classes, FDA only has $k-1$ solutions, therefore 3 subplots in LDA plot. On the LDA plot, we can see clearly the separation between classes along each LDA axis and the cohesion within each class because these are the optimization criteria of LDA. But we cannot see the same pattern in the PCA plot as PCA only maximize the overall variance on each PCA axis.