CSE-343 – Machine Learning

Professor : Dr Jainendra Shukla

Report by Deepanshu Dabas(2021249)

Section A

Ans a)

a)

1. The output of the Convoluation layer is given by W2 = (W1 − F + 2P)/S + 1 and H2 = (H1 − F + 2P)/S + 1
2. The output of the pooling layer is given by W2 = (W1 − F)/S + 1 and H2 = (H1 − F)/S + 1

   From 1 and 2,
   - 1st Convoluation layer Output is: 13 x13
   - Pooling Layer Output is: 6 x 6
   - Last Layer Output is 3.5 x 4.5

   Round of last layer output is 4 x 5

   -> Hence output image size is 4 x 5.

b) Pooling in Convolutional Neural Networks (CNNs) plays a vital role due to following reasons:

1.**Translation Invariance:** Pooling helps the model to recognize patterns irrespective of their position in the image. This means that even if the pattern shifts or moves within the image, the pooled feature map will still be able to detect it. This property is known as translation invariance.

2.**Noise and Distortion Reduction:** Pooling layers effectively reduce the spatial dimensions of the input, which not only decreases the computational complexity but also controls overfitting by providing an abstracted form of the representation. As a result, the model becomes less sensitive to noise and distortions in the input image.

c) The learnable parameters of a convolutional layer in neural network are given by the formula: $i \times o \times m \times n$

where:

- i is the number of input channels,
- o is the number of output channels,
- m and n are the dimensions of the filter.

-> Pooling layers do not have any learnable parameters.

Therefore ,the learnable parameters in each layer are:

1. First Layer: The learnable parameters are calculated as 5×5×4×1=100.
2. Second Layer: As this is a pooling layer, it has 0 learnable parameters.
3. Third Layer: The learnable parameters can be calculated as 5×3×4×1=60

Therefore, the total number of parameters in the network would be 100+60=160.

Ans b) No, the k-means algorithm doesn’t revisit a configuration once it has moved on from it. K means algorithm in each iteration assigns each instance to the cluster whose centroid is nearest based upon loss function. It then recalculates the centroids based on the new assignments. This process is repeated until the assignments no longer change between iterations.
Since each iteration reduces the sum of squared distances or increases silhouette score from instances to their assigned centroids (the objective function), and there are only a finite number of possible configurations, the algorithm cannot revisit a configuration. If it did, it would create a loop and never terminate, contradicting the assumption that the objective function of kmeans algorithm always decreases. Hence k means algorithm converges in a finite number of steps.

Ans c)Yes, a neural network can be used to model K-Nearest Neighbours algorithms.It’s theoretically possible to model KNN with a neural network, it may not always be the most efficient or practical approach. We can express K-NN as 3 layer neural network consisting of input layer, one hidden layer consisting of RBF activation function ,each neuron in the hidden layer represents one instance of the training data and calculates the Euclidean distance from the input vector to the corresponding training instance and one output layer consisting of linear function. The centers of the RBFs can be set to the k nearest neighbors of each data point in the training set. The weights of the RBFs can then be set to the distances between the data points and their k nearest neighbors. This RBF network will approximate the behavior of the KNN algorithm. The accuracy of the approximation will depend on the number of RBFs used and the way that the centers and weights of the RBFs are set.

Ans d) Linear kernels are able to catch only near relationships between data points. This simply means that they are limited to detecting patterns that can be represented by straight lines. Hence, they are computationaly efficient and are generally used for simple tasks.Example of linear kernel is box filter .On the other hand, non linear kernels are able to catch complex pattern between datapoints and therefore generally used in complex tasks such as image classification, object detection,etc. They are computationaly expensive and require lot of resource . Example of non linear kernels are Sigmoid function, gaussian filter,etc.

Section B

Convolution Functions

•**Convolution Forward:** This function applies a special operation called convolution to an image. It uses a small matrix called a kernel. To make sure we don't lose information from the edges of the image, we add some extra pixels around the image, a process known as zero padding. This function is a key part of how a Convolutional Neural Network (CNN) works, helping it to find patterns in images.

•**Convolution Backward:** This function helps us improve our model. It takes in information about how far off our model's predictions were (the gradients of the loss from the pooling layer) and uses this to adjust the values in the kernel, aiming to make our model's predictions better.

•**Window:** This function creates a kernel with random values. The size of the kernel is specified as an input.

•**Zero Padding:** This function adds extra pixels around the edge of the image. This is done so that when we apply the kernel to the image, we can also apply it to the pixels at the edge of the image.

Pooling Functions

•**Create_Mask:** This function is used during the backpropagation process, which is how our model learns from its mistakes. It creates a mask, which is a way of remembering where the most important information came from in the original image. This helps us know which parts of the image we should pay more attention to when we're adjusting our model.

•**Distribute:** This function spreads out gradients to all the pixels in an image that contributed to a value in an average pooled image. This is because in an average pooled image, each pixel is the average of a group of pixels from the original image.

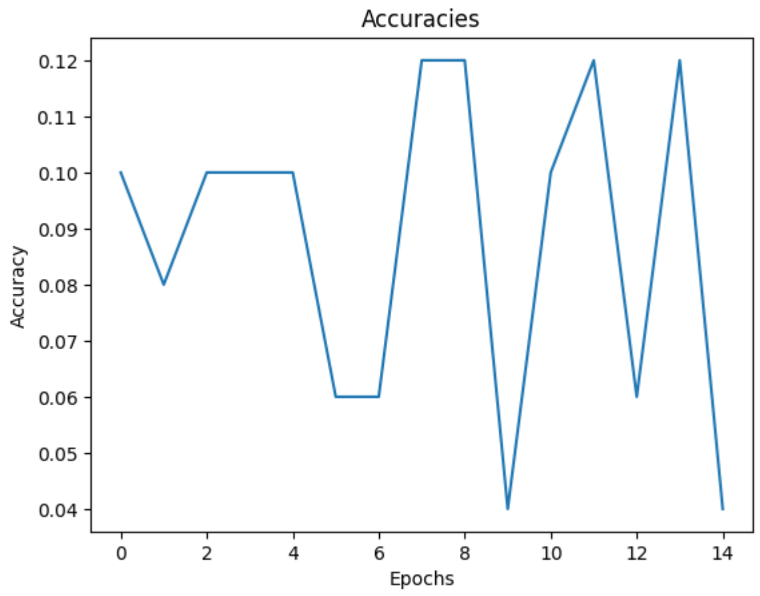
•**Pooling Forward:** This function applies a pooling operation to the image. Pooling is a way of reducing the size of the image, while keeping the most important information. Depending on the type of pooling specified, it applies either MAX pool or AVG pool.

•**Pooling Backward:** This function chooses the right method to send the image back to the convolution layer, depending on the type of pooling operation that was performed in the feed-forward phase.

Code Logic:

1. **Data Loading:**
    - The code uses the mnist library to load the MNIST dataset, which consists of handwritten digit images.
2. **Convolution Operation:**
    - The Convolution class contains a method convolve that performs the convolution operation on an input image with a given kernel (filter).
    - The convolution operation involves applying a filter to the input image, element-wise multiplication, and summing up the results. This is done for each position in the input image to generate a convolved feature map.
    - The backpropagation is also implemented (backprop method) to update the kernel weights based on the gradient of the loss with respect to the convolved output.
3. **Pooling Layer:**
    - The Pooling class includes methods for both max pooling (pooling) and average pooling (avg_pooling).
    - Pooling is performed to reduce the spatial dimensions of the convolved feature map while retaining important information. Max pooling takes the maximum value in each pooling window, while average pooling takes the average.
    - The create_mask method creates a mask to distribute gradients during backpropagation for max pooling.
    - The distribute method distributes the gradients for average pooling.
4. **Activation Layer:**
    - The ActivatedLayer class represents the fully connected layer with softmax activation.
    - The softmax_layer method applies the softmax activation function to the input and calculates the output probabilities.

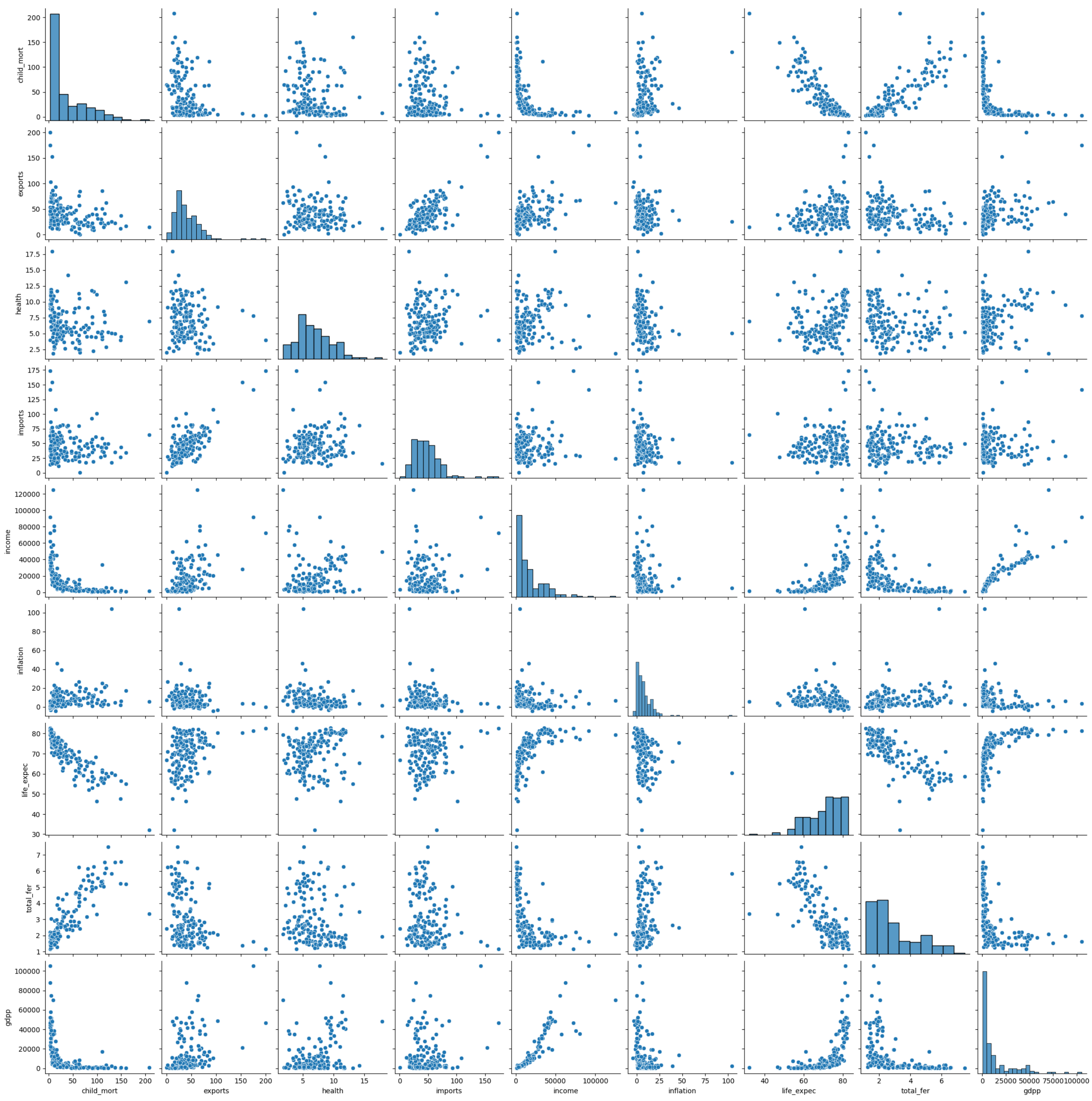
    - The backprop method performs backpropagation to update the weights and biases based on the calculated gradient of the loss with respect to the output.
5. **Training Loop:**
    - The code includes a training loop with a fixed number of epochs.
    - For each epoch, the model is trained on a subset of the MNIST test set.
    - The accuracy and log loss are computed during training.
6. **Visualization:**
    - The code includes a simple plot of log losses over epochs.



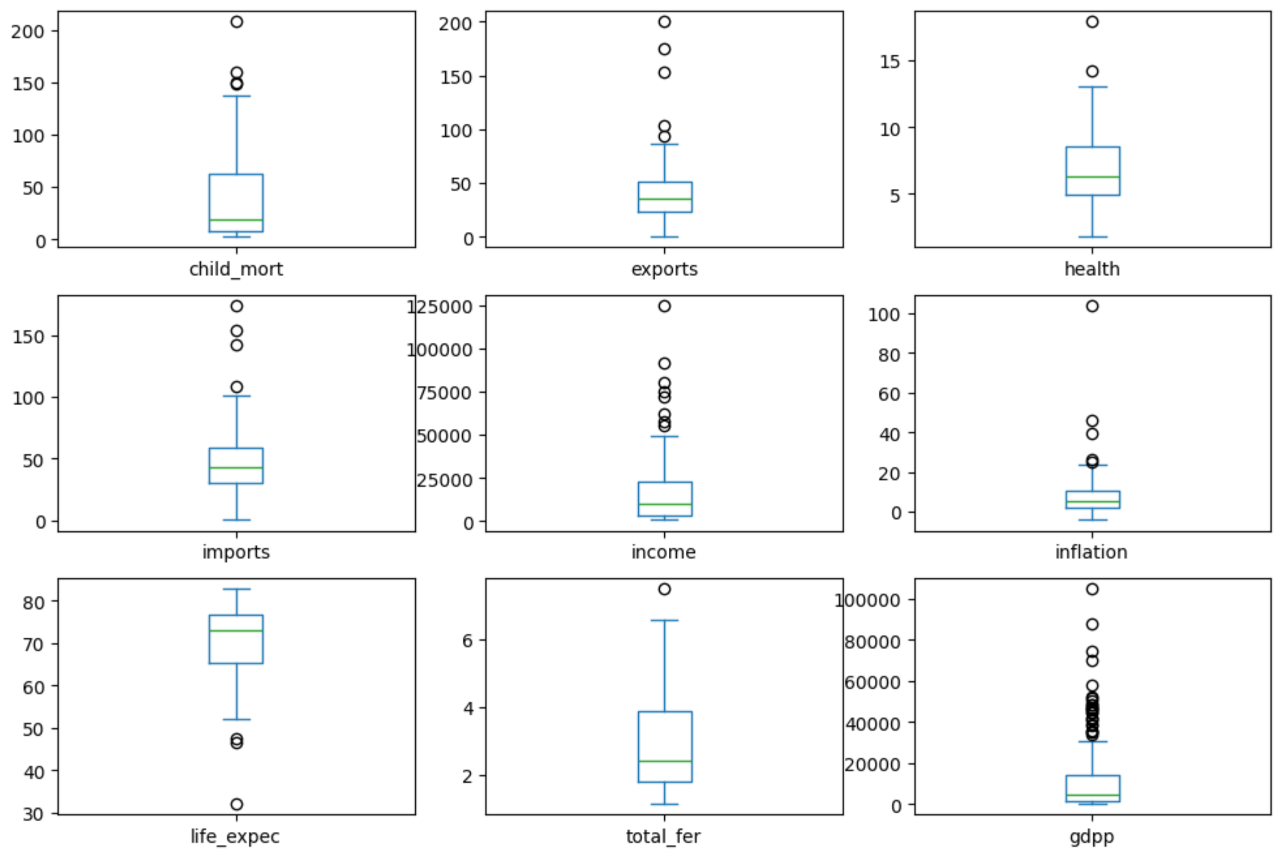
Miscellaneous:
- Random initialization of kernels and other parameters hence inconsistent results for re-run of code.
- Learning rate and other hyperparameters are manually set.

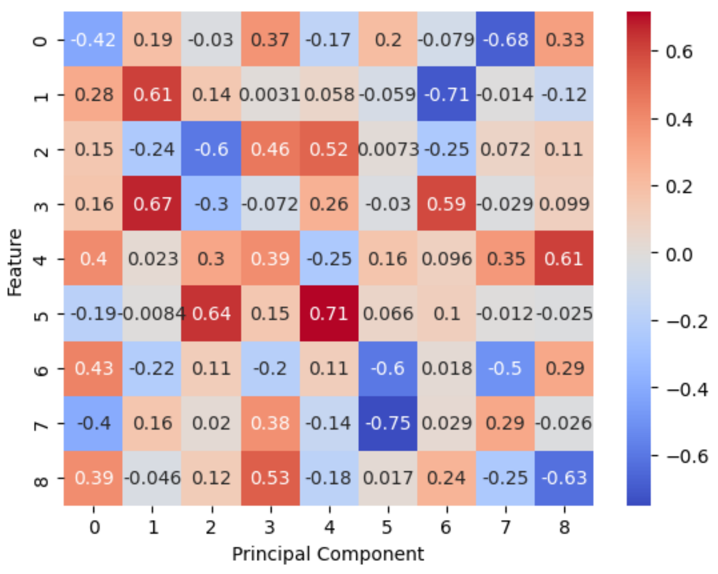
Section C

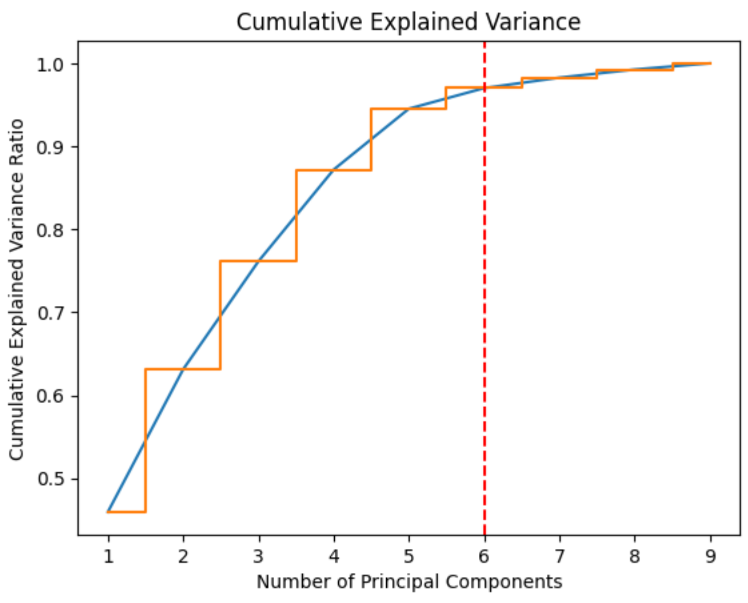
code_2021249_Section_C.ipynb notebook is markdown with all the necessary comments, you can check that also. Attaching some screenshot from there here also:
please refer notebook for code and logic as it consist of every plot(scatter plots are many and need hardwork to add in report,Please refer notebook for better analysis)
Pair Plot:

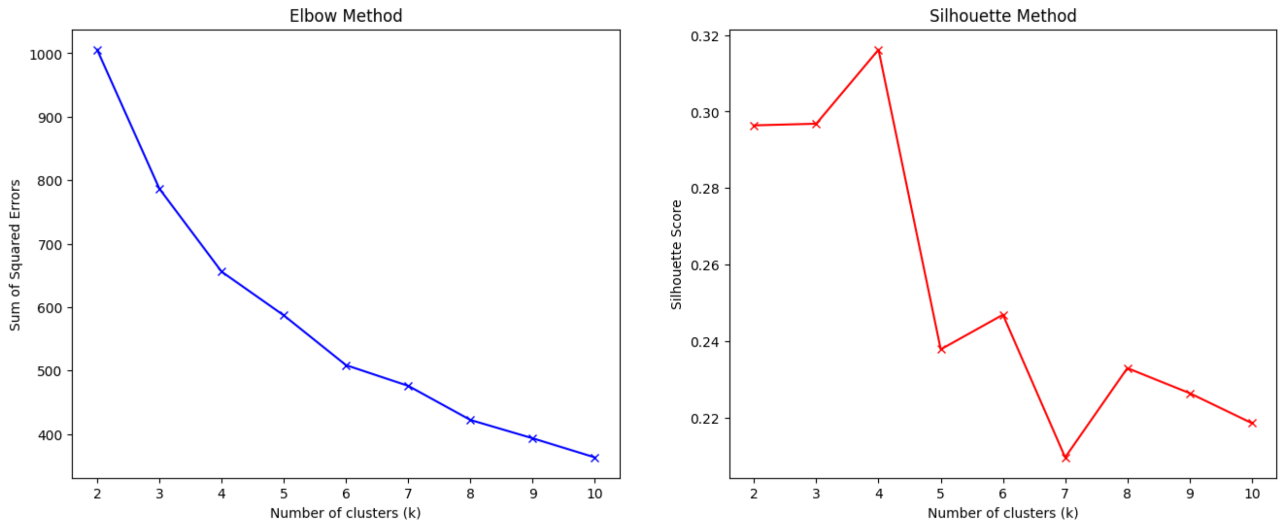


child_mort
exports
health
imports
income
inflation
life_expec
total_fer
gdpp

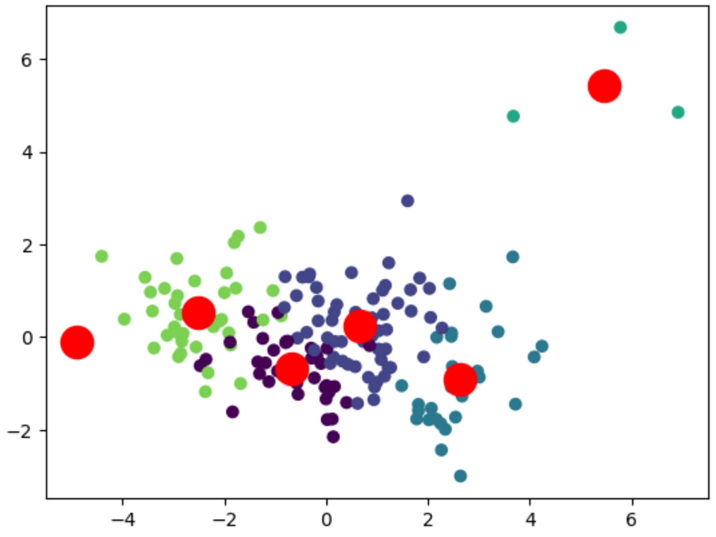

child_mort
exports
health
imports
income
inflation
life_expec
total_fer
gdpp

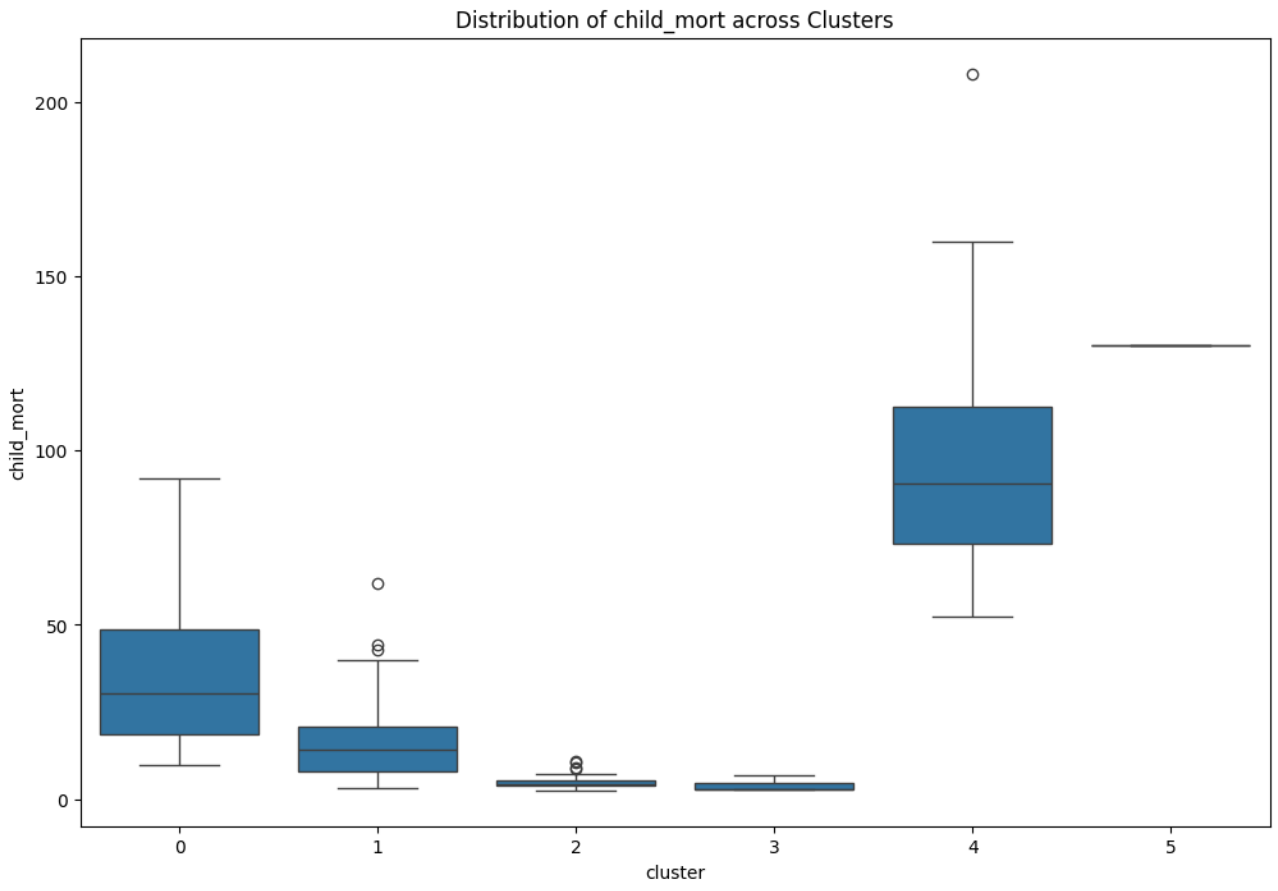

Feature
Principal Component
0: -0.42 0.19 -0.03 0.37 -0.17 0.2 -0.079 -0.68 0.33
1: 0.28 0.61 0.14 0.0031 0.058 -0.059 -0.71 -0.014 -0.12
2: 0.15 -0.24 -0.6 0.46 0.52 0.0073 -0.25 0.072 0.11
3: 0.16 0.67 -0.3 -0.072 0.26 -0.03 0.59 -0.029 0.099
4: 0.4 0.023 0.3 0.39 -0.25 0.16 0.096 0.35 0.61
5: -0.19 -0.0084 0.64 0.15 0.71 0.066 0.1 -0.012 -0.025
6: 0.43 -0.22 0.11 -0.2 0.11 -0.6 0.018 -0.5 0.29
7: -0.4 0.16 0.02 0.38 -0.14 -0.75 0.029 0.29 -0.026
8: 0.39 -0.046 0.12 0.53 -0.18 0.017 0.24 -0.25 -0.63


Cumulative Explained Variance
Cumulative Explained Variance Ratio
Number of Principal Components


Elbow Method
Sum of Squared Errors
Number of clusters (k)
Silhouette Method
Silhouette Score
Number of clusters (k)

Clusters:




Distribution of child_mort across Clusters
child_mort
cluster