Shourya Jindal
2020336
ML Assignment 4

Section A

(Q1)(a)(a)

Input Image = 15 x 15 x 4

1. First ConVo 2D: Kernel = 5 x 5 x 4 x 1
Padding = 1
Stride = 1

$$\text{Output} = \frac{\text{input} - \text{kernel} + 2 \times \text{padding}}{\text{stride}} + 1$$

$$\text{So, output height \& width} = \frac{15 - 5 + 2(1)}{(1)} + 1$$

$$= 13$$

and output channels = No. of o/p channels from ConVo2D = 1

So, **o/p = 13 x 13 x 1**

2. Max Pooling Layer: Kernel = 3 x 3
Stride = 2

$$\text{output} = \frac{\text{input} - \text{kernel} + 2 * \text{padding}}{\text{stride}} + 1$$

$$= \frac{13 - 3 + 0}{2} + 1$$

$$= 6$$

So, **o/p = 6 x 6 x 1**

3. ConVo2D Kernel = 5 x 3 x 4 (1)

Padding = 2

Stride = 2

Input from last layer: 6 x 6 x 1

$$\text{Output} = \frac{\text{Input} - \text{Kernel} + 2 * \text{padding}}{\text{Stride}} + 1$$

$$\text{Output height} = \frac{6 - 5 + 2(2)}{2} + 1$$

$$= 3.5 \rightarrow \text{taking floor} = \boxed{3}$$

$$\text{Output width} = \frac{6 - 3 + 2(2)}{2} + 1$$

$$= 4.5 \rightarrow \text{taking floor} = \boxed{4}$$

Opp channels = o/p channels of convo 2D = 1

So, **final output = 3 x 4 x 1** → Ans

(Q1) (a) (b) Pooling has crucial roles in CNN:

- Dimensionality Redn: It reduces the spatial dimensions (W & H) of the input, thus reducing no. of parameters and also reducing computational load.
- Translation Invariance: It helps create a degree of translation invariance by summarizing the presence of features within small region.
- Feature Generalization: It helps in generalizing learned features. It helps capture most relevant features and discard less relevant feature
- It reduces overfitting by introducing a type of regularization.
- Increases computational efficiency by reducing W & H of i/p.

(Q1) (a) (c) Ignoring Bias

Total learnable parameters:

First Conv2D → Kernel: 5×5×4×1 : no. of learnable parameters = 5×5×4×1

= 100

Max pooling : parameters = 0

2nd conv2D : Kernel = 5×3×4×1

no. of learnable parameters = (5×3×4 (from pr. layer) ×1) ×1

= 60

So, total learnable parameters (w/o bias) = 100 + 0 + 60

= 160 → Ans

(Q1) (b)

No, It is not possible for the k-means algo. to revisit a configuration.

If it revisits a configuration during an iteration then, k-means may not always converge.

The converging condition is that : k-way partition does not changes in successive iteration.

Also, since the mean squared error and the loss func^n monotonically decreases with successive iterations, it is impossible to revisit a configuration

There are finite no. of k-way partitions possible, thus it must converge in finite steps. And to converge, it must not revisit a config. If it does, it will remain same in subsequent iterations and you have found the final clusters.

For Eg: $n = 4$ (total points)

$K = 2$ (total clusters)

let data pts be : ① ② ③ ④

Clusters possible: $\{(1,2),(3,4)\} \rightarrow A$

$\{(1,3),(2,4)\} \rightarrow B$

$\{(1,4),(2,3)\} \rightarrow C$

with error as following $\quad A \; B \; C$ (decreasing error)

If iteration starts from (A) then go to (B), then it will never return to (A), it will always go to (C).

(Q1)(c) No, a neural network can not be used to model K-NN algorithms.

This is because:

- KNN is an instance-based i.e instance-matching learning algorithm. There is no as such 'learning' in the model. It just stores the entire training dataset and uses it directly for prediction.
- Also, KNN do not have any learning parameters that are needed to be trained

It just predicts the y-label by find 'k' nearest data points using Euclidean distance.

This is in contrast to what a neural network does. It involves training on data, parameter learning (iteratively).

(Q1)(d)

Linear Kernels	Non-Linear Kernels
• It performs linear transformation on the input data.	• It performs non-linear transformation on the input data.
• They captures linear relationships between the input features	• They captures non linear complex relationships b/w the input features, that linear kernel may not capture. It can also handle input data i.e not linearly seperable
• It may not handle data i.e not linearly seperable.	• non linear transformations
• series of linear models	• can capture non-linearities
• cannot capture non-linearities	• can also handle complex data with high dimensions which may not be handled by linear kernels
Eg: matrix multiplication, addition identity func^n.	Eg: ReLU, Sigmoid

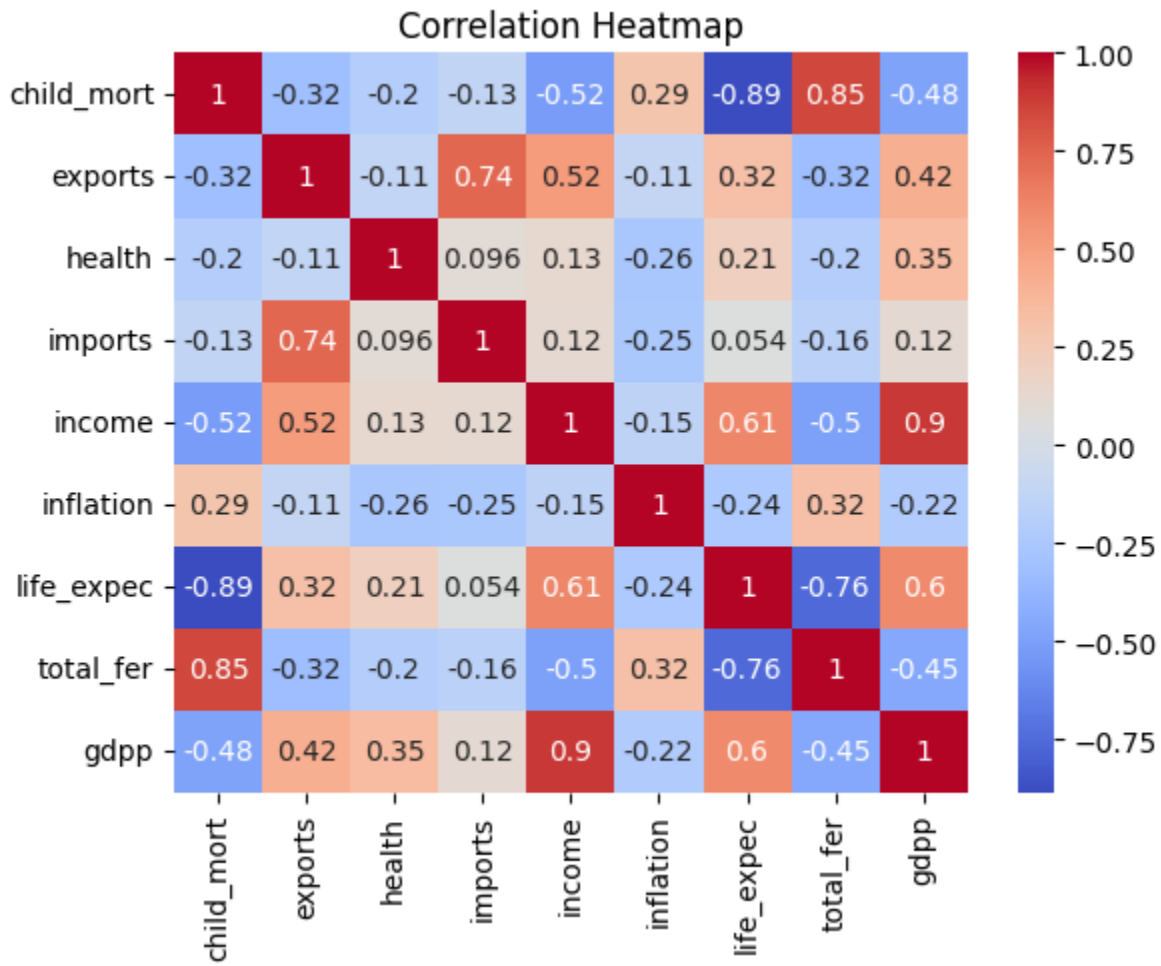
SECTION C

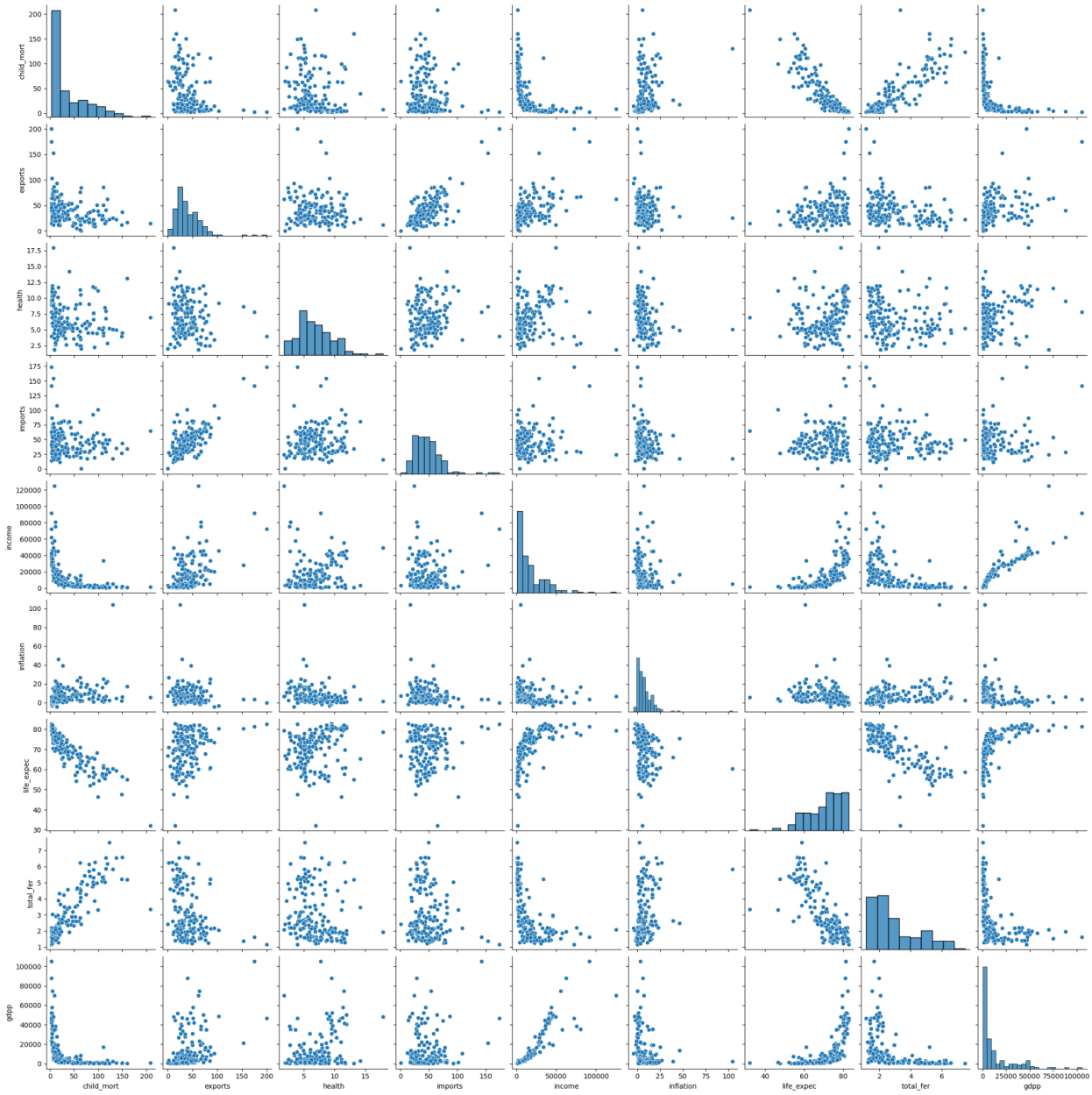
Part A: EDA and Preprocessing:

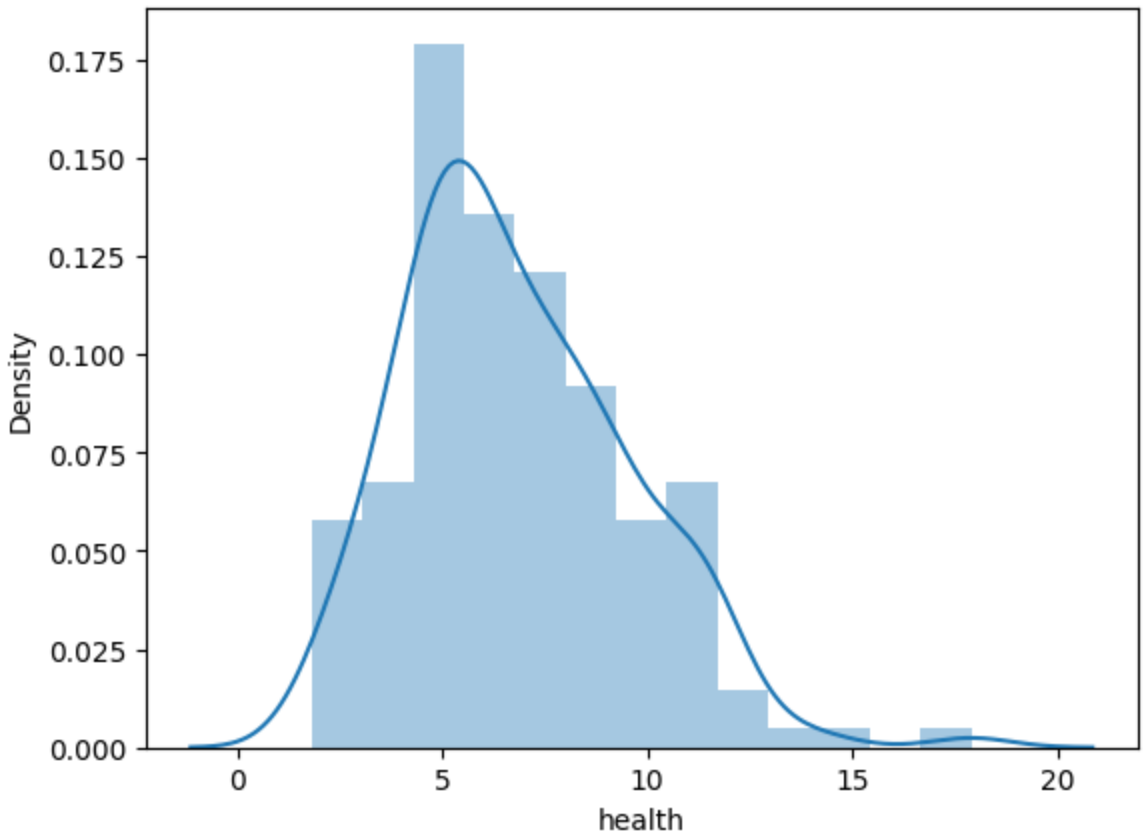
Loaded the dataset:
Rows: 167
Columns: 10: 1 country column and 9 describing that country's features.

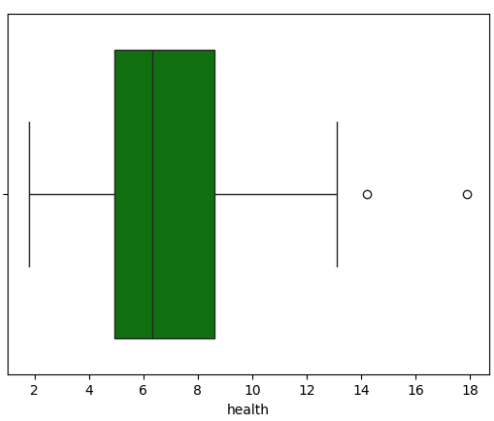
Checked for null or missing values, found none.

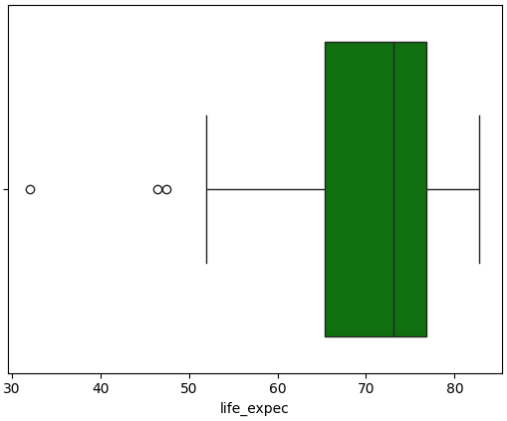
Did EDA by making numerous plots and using describe() to find more about the given features like mean, median, mode etc. Also plotted boxplots to find if there are any outliers. Some are:




child_mort
exports
health
imports
income
inflation
life_expec
total_fer
gdpp


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
0
5
10
15
20
health


2
4
6
8
10
12
14
16
18
health


30
40
50
60
70
80
life_expec

Some insights:

- GDP is positively correlated with income, life expectancy, and exports and negatively correlated with child mortality and inflation.
- Child mortality is positively correlated with inflation. As inflation rises, economic conditions worsen, leading to high child mortality.
- A high value of health features leads to high life expectancy and lower child mortality.
- The number of outliers found using boxplots is much less, approximately 2-3 for each feature. Since the data only has 167 countries, we do not remove them.

Preprocessing:

No null or missing values were found.

Used Standard Scaler from Sklearn, to standardize the features by scaling technique.

Part B: PCA

Used PCA from sklearn to reduce the dimensionality of the data.

```
from sklearn.decomposition import PCA
pca = PCA()
pca1 = pca.fit_transform(df_scaled)
cols = []
for i in range(1, 10):
    cols.append('PC'+str(i))
pca1 = pd.DataFrame(pca1, columns=cols)
pca1
```

✓ 0.0s

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
0	-2.913025	0.095621	-0.718118	1.005255	-0.158310	-0.254597	0.383000	0.415076	-0.014148
1	0.429911	-0.588156	-0.333486	-1.161059	0.174677	0.084579	0.248919	-0.221042	0.173316
2	-0.285225	-0.455174	1.221505	-0.868115	0.156475	-0.401696	-0.087214	-0.184162	0.084037
3	-2.932423	1.695555	1.525044	0.839625	-0.273209	-0.547996	-0.440835	-0.355998	-0.091339
4	1.033576	0.136659	-0.225721	-0.847063	-0.193007	-0.206919	0.241978	-0.023681	0.094270
...	...	...	...	...	...	...	...	...	...
162	-0.820631	0.639570	-0.389923	-0.706595	-0.395748	0.009059	-0.098738	0.521886	-0.497803
163	-0.551036	-1.233886	3.101350	-0.115311	2.082581	0.097274	0.206735	-0.078805	-0.275735
164	0.498524	1.390744	-0.238526	-1.074098	1.176081	0.051607	-0.143627	-0.217590	-0.036522
165	-1.887451	-0.109453	1.109752	0.056257	0.618365	-0.540917	0.060256	0.089495	-0.096049
166	-2.864064	0.485998	0.223167	0.816364	-0.274068	0.201378	-0.442185	0.664338	-0.441482

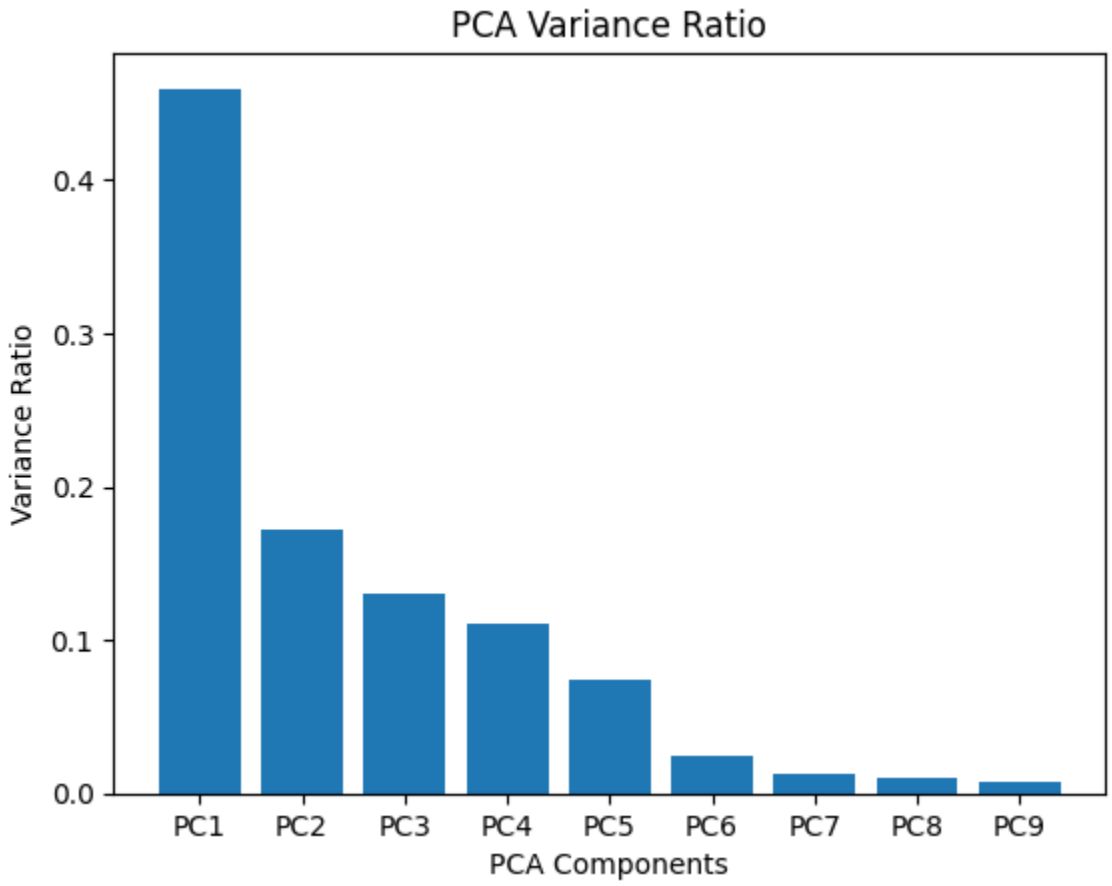
167 rows × 9 columns

To find the optimal number of principal components used explained variance ratio.

[0.4595174 0.17181626 0.13004259 0.11053162 0.07340211 0.02484235 0.0126043 0.00981282 0.00743056]

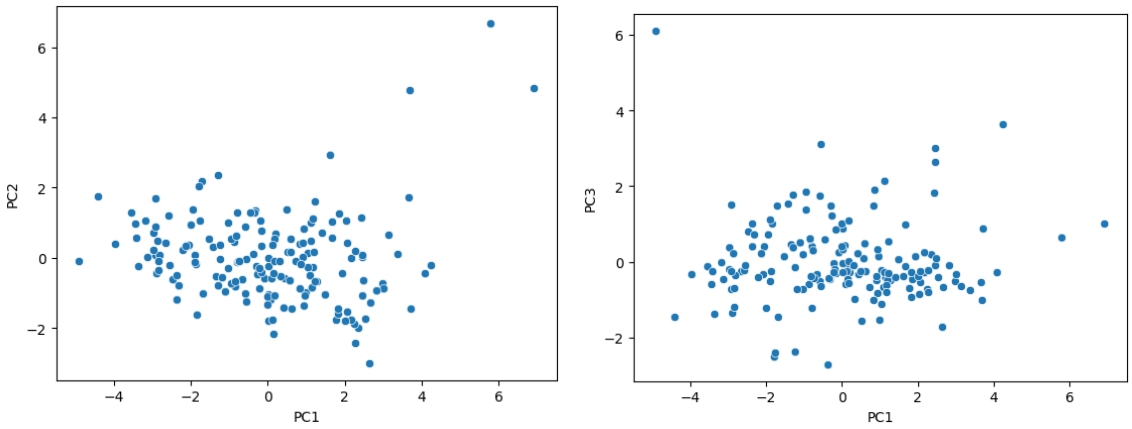
These were the values that came out.

Plot:

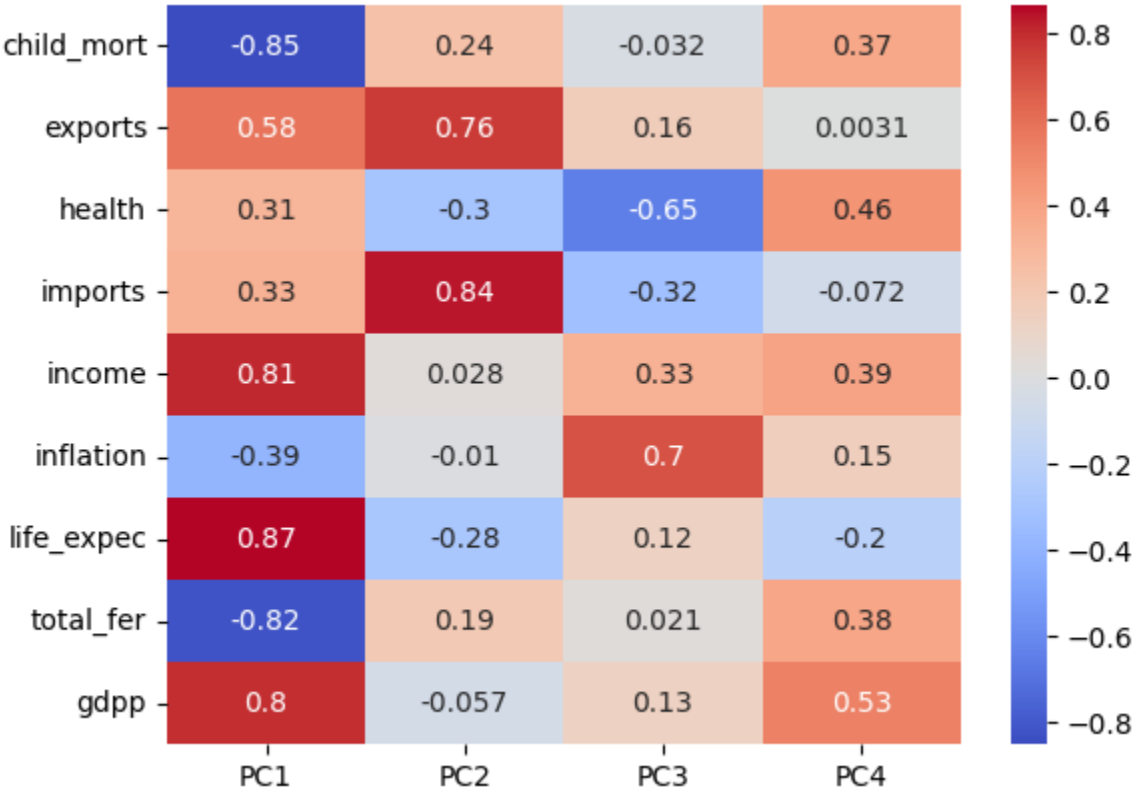


From this, I chose the **optimal number of principal components to be = 4**. This is because it retained approximately 90% of the variance, which was enough for correct analysis.

Heatmaps and scatter plots to visualize these PCAs:



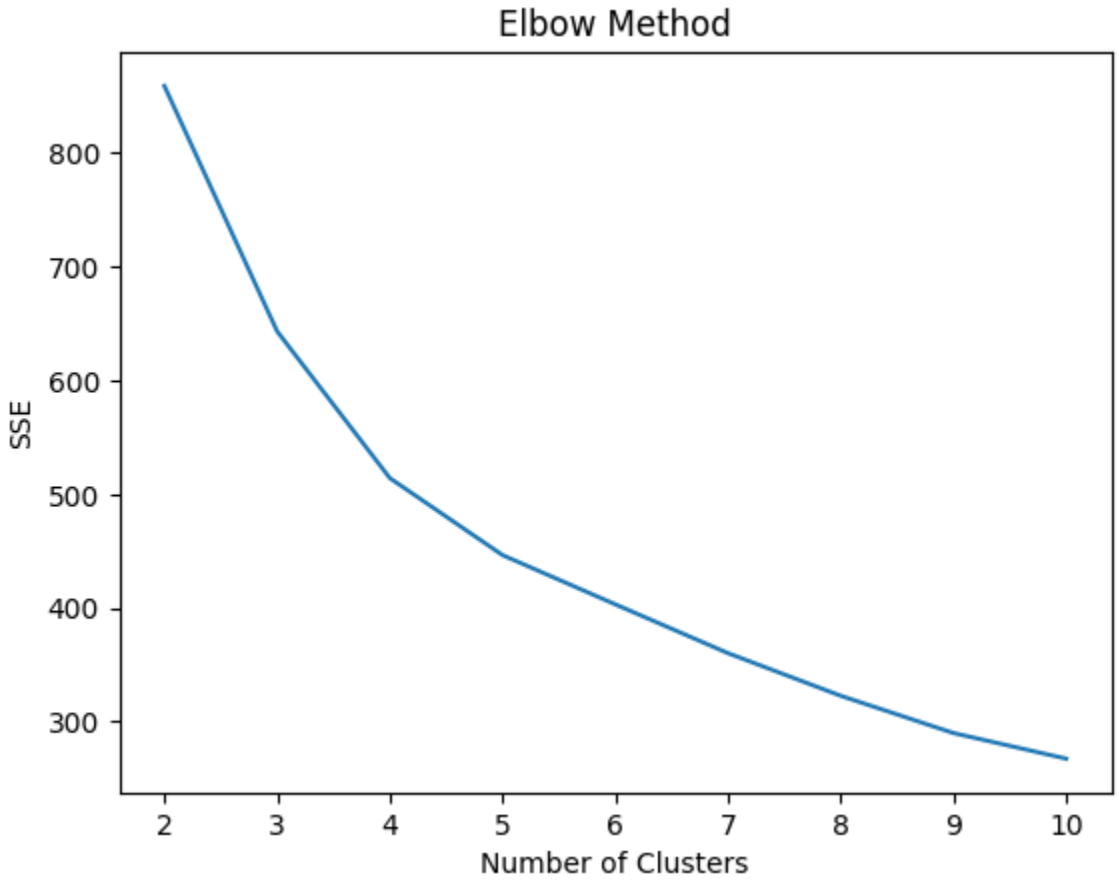
Heatmaps showing the PCAs and their correlation with the original features:

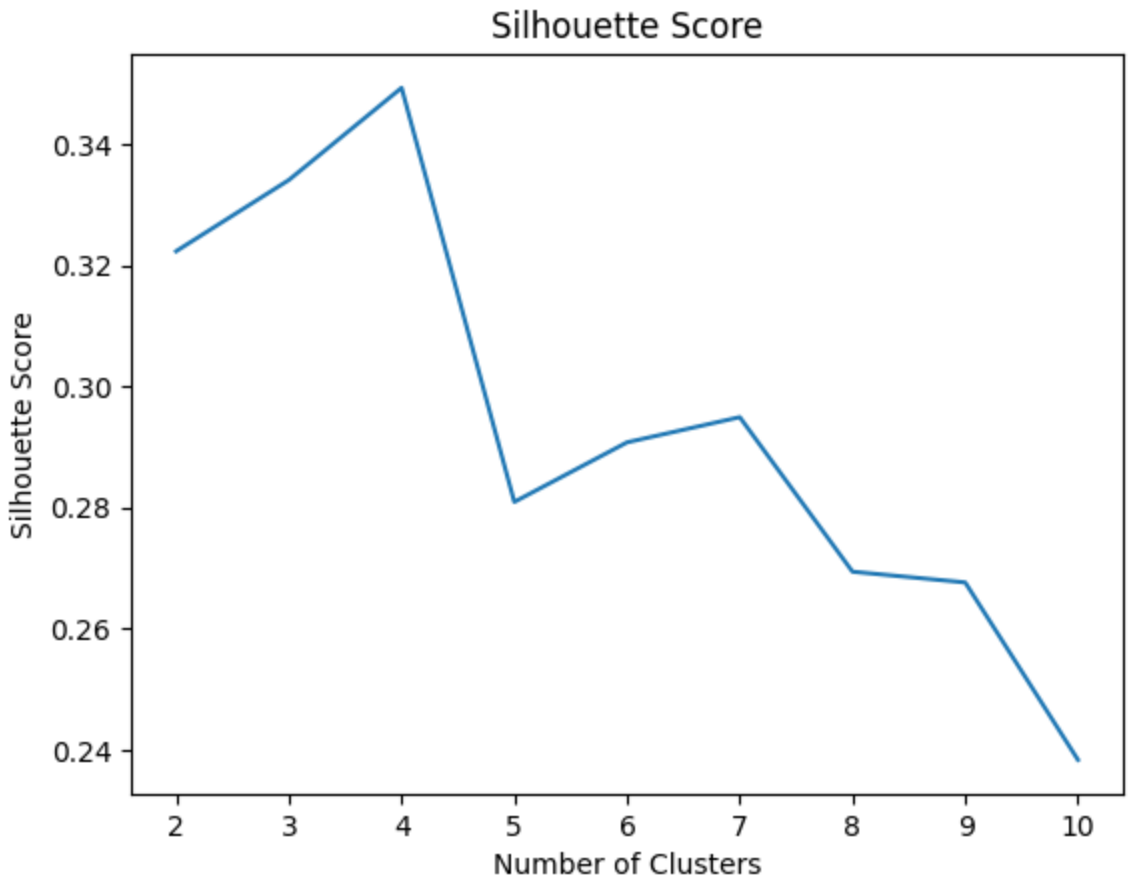


PART C: K-MEANS

On this scaled and PCA and transformed data, I applied K-MEANS clustering.

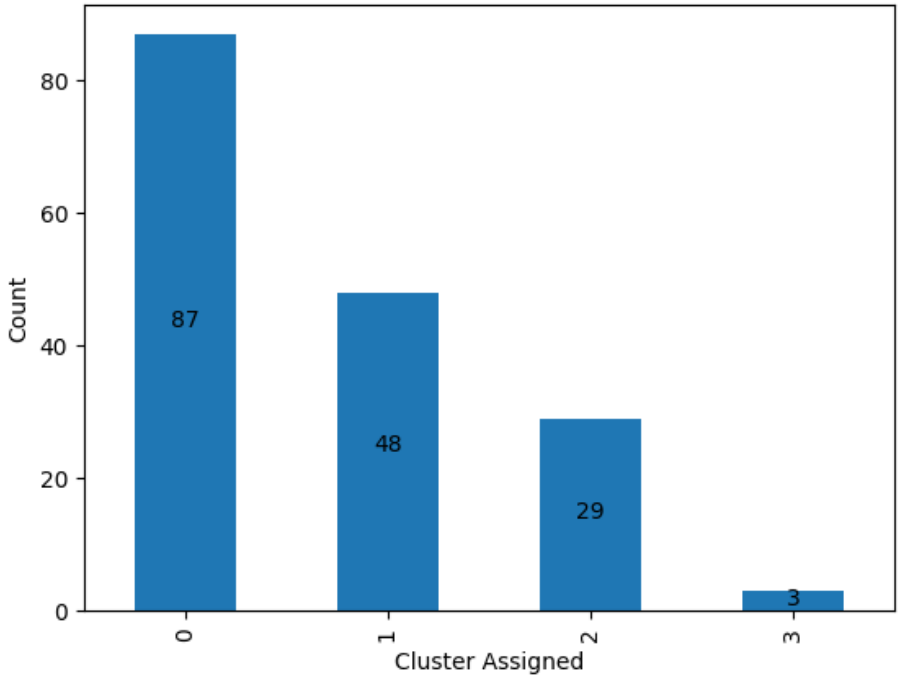
First, to find the number of optimal clusters I used the Elbow-curve method and silhouette score method. Plots:



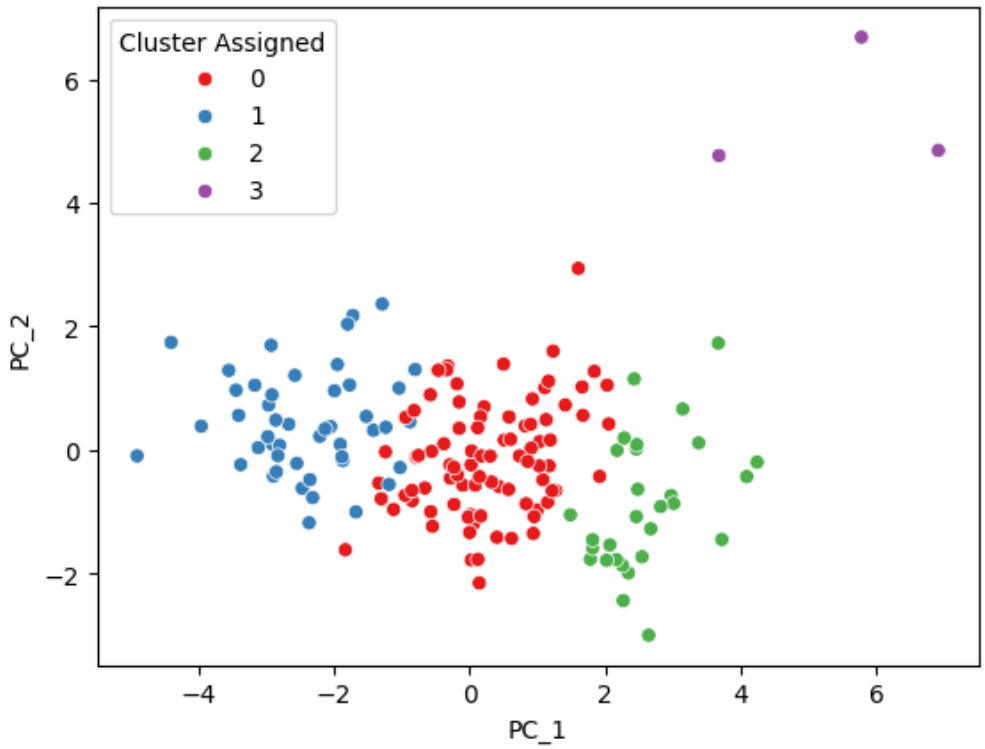


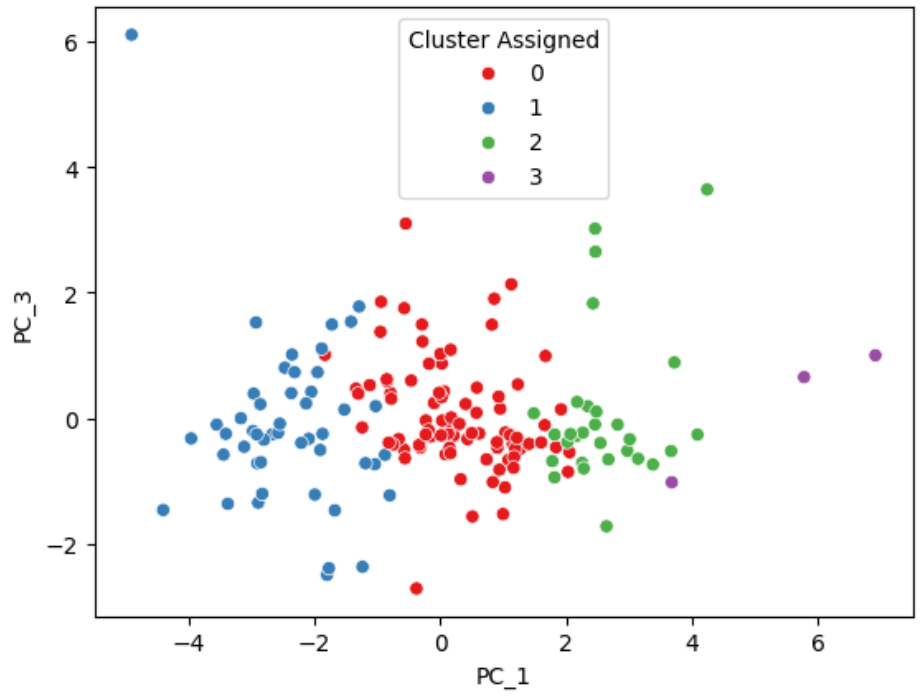
From both the methods, optimal clusters came out to be = 4.

Then using k = 4, assigned cluster labels to the data of countries. Count of each cluster came out to be:



To do further analysis of the clusters many plots were used. Some of them are:
2D Scatter Plots:





Analysed the features of each cluster:

```
#Cluster 0
df[df['Cluster Assigned'] == 0].describe()
```

✓ 0.0s

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Assigned
count	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000	87.0
mean	21.113793	41.291828	6.217241	47.636390	13297.586207	7.371506	73.147126	2.251954	7158.298851	0.0
std	13.925092	19.523609	2.145495	19.759309	8980.625783	7.821240	3.829653	0.672577	5656.749026	0.0
min	3.400000	0.109000	1.970000	0.065900	1990.000000	-4.210000	63.000000	1.230000	592.000000	0.0
25%	10.700000	26.900000	4.885000	32.700000	6720.000000	2.320000	70.400000	1.670000	2980.000000	0.0
50%	17.400000	37.700000	5.980000	48.700000	10900.000000	5.710000	73.900000	2.170000	5080.000000	0.0
75%	26.900000	52.050000	7.395000	60.550000	17900.000000	9.905000	76.200000	2.645000	10700.000000	0.0
max	64.400000	93.800000	14.200000	108.000000	45400.000000	45.900000	80.400000	4.340000	28000.000000	0.0

```
#Cluster 1
df[df['Cluster Assigned'] == 1].describe()
```
✓ 0.0s

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Assigned
count	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.0
mean	91.610417	29.571042	6.433542	43.133333	3897.354167	11.911146	59.239583	4.992083	1909.208333	1.0
std	34.319855	18.200215	2.651959	18.418658	5590.168621	15.362485	6.384914	1.036192	2925.911009	0.0
min	28.100000	2.200000	2.200000	17.200000	609.000000	0.885000	32.100000	2.590000	231.000000	1.0
25%	63.675000	17.025000	4.525000	29.900000	1390.000000	4.080000	56.725000	4.475000	551.500000	1.0
50%	89.750000	24.350000	5.675000	41.500000	1860.000000	8.855000	59.800000	5.055000	932.000000	1.0
75%	111.000000	39.400000	8.327500	50.025000	3522.500000	16.600000	62.825000	5.597500	1465.000000	1.0
max	208.000000	85.800000	13.100000	101.000000	33700.000000	104.000000	71.100000	7.490000	17100.000000	1.0

```
#Cluster 2
df[df['Cluster Assigned'] == 2].describe()
```
✓ 0.0s

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Assigned
count	29.000000	29.000000	29.000000	29.000000	29.000000	29.000000	29.000000	29.000000	29.000000	29.0
mean	4.982759	45.703448	9.245862	39.513793	45762.068966	2.727793	80.386207	1.814828	44065.517241	2.0
std	2.191227	22.110332	3.296145	17.720604	19932.178108	4.341140	1.464758	0.359778	14752.173590	0.0
min	2.600000	12.400000	1.810000	13.600000	27200.000000	-3.220000	76.500000	1.370000	22500.000000	2.0
25%	3.800000	28.200000	8.950000	28.000000	35800.000000	0.643000	79.800000	1.480000	35000.000000	2.0
50%	4.200000	42.300000	9.540000	32.900000	40700.000000	1.160000	80.400000	1.870000	41900.000000	2.0
75%	5.200000	64.000000	11.000000	47.800000	45700.000000	3.220000	81.400000	1.980000	48700.000000	2.0
max	10.800000	103.000000	17.900000	86.500000	125000.000000	16.700000	82.800000	3.030000	87800.000000	2.0

```
#Cluster 3
df[df['Cluster Assigned'] == 3].describe()
```
✓ 0.0s

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Assigned
count	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.0
mean	4.133333	176.000000	6.793333	156.666667	64033.333333	2.468000	81.433333	1.380000	57566.666667	3.0
std	2.309401	23.515952	2.492877	16.165808	32460.642836	2.179718	1.205543	0.240624	43011.665084	0.0
min	2.800000	153.000000	3.960000	142.000000	28300.000000	-0.046000	80.300000	1.150000	21100.000000	3.0
25%	2.800000	164.000000	5.865000	148.000000	50200.000000	1.787000	80.800000	1.255000	33850.000000	3.0
50%	2.800000	175.000000	7.770000	154.000000	72100.000000	3.620000	81.300000	1.360000	46600.000000	3.0
75%	4.800000	187.500000	8.210000	164.000000	81900.000000	3.725000	82.000000	1.495000	75800.000000	3.0
max	6.800000	200.000000	8.650000	174.000000	91700.000000	3.830000	82.700000	1.630000	105000.000000	3.0

After analyzing these data, concluded that:

Following features: GDP, Income, life expectancy, exports, and health have the following order in clusters: 3 > 2 > 0 > 1

And features: child mortality, and inflation have the following order: 3 < 2 < 0 < 1.

Cluster 0: Developing
Cluster 1: Under-Developed
Cluster 2: Developed
Cluster 3: Highly-Developed

Assigned these categories of clusters to the original data.