Final project report

Step 1:

For the first step of the assignment, I selected the Haberman's Survival dataset from the UCI Machine Learning Repository. The dataset presents the survival status of patients who underwent breast cancer surgery between 1958 and 1970, based on attributes like age at the time of operation, the year of operation, and the number of positive axillary nodes detected.

I started off by importing a range of Python libraries that I'd be needing for various tasks throughout this assignment, like data handling, visualization, and model building and evaluation.

Next, I loaded the dataset directly from the UCI repository using the provided URL and made the data more understandable by assigning appropriate column names.

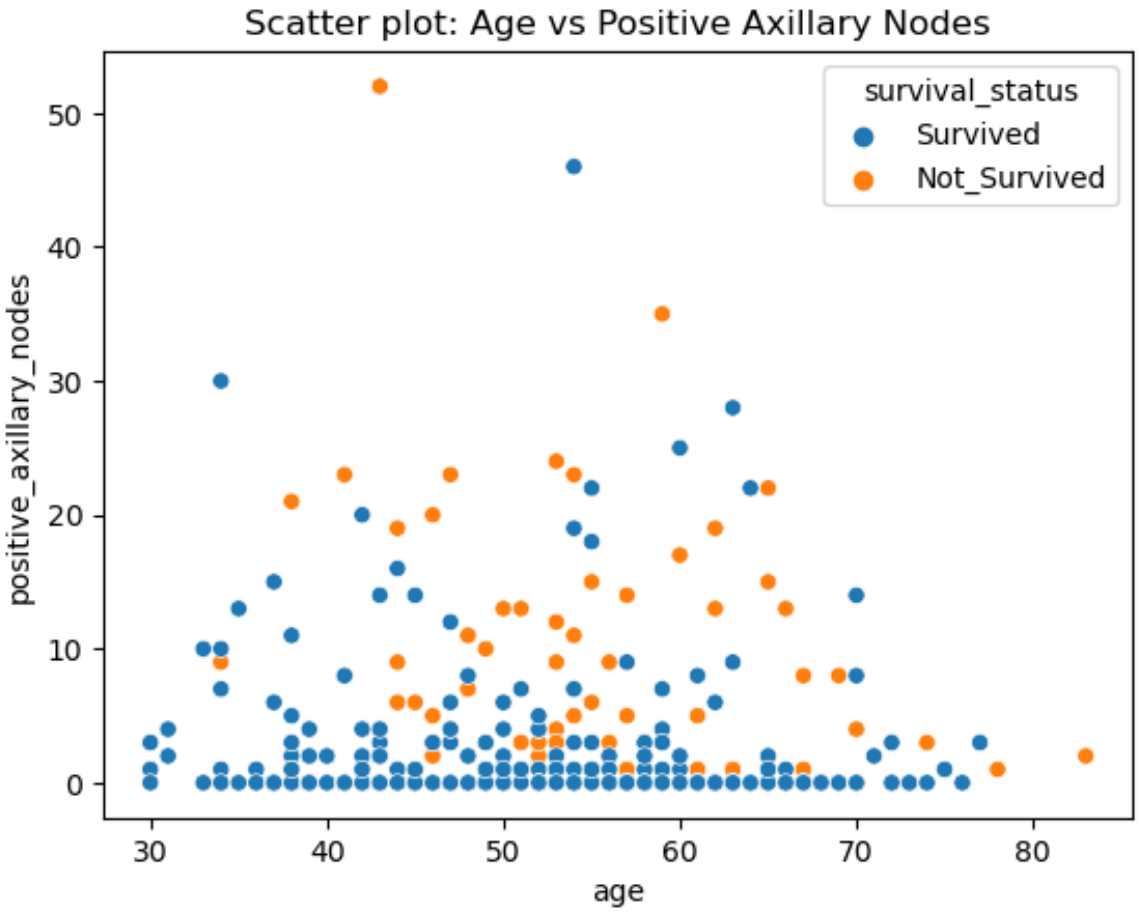
To gain a basic understanding of my dataset, I printed the shape, previewed the first few entries using the head() function, checked the data types of each column using info(), and obtained a statistical summary using describe().

	age	year_of_operation	positive_axillary_nodes	survival_status
0	30	64	1	1
1	30	62	3	1
2	30	65	0	1
3	31	59	2	1
4	31	65	4	1

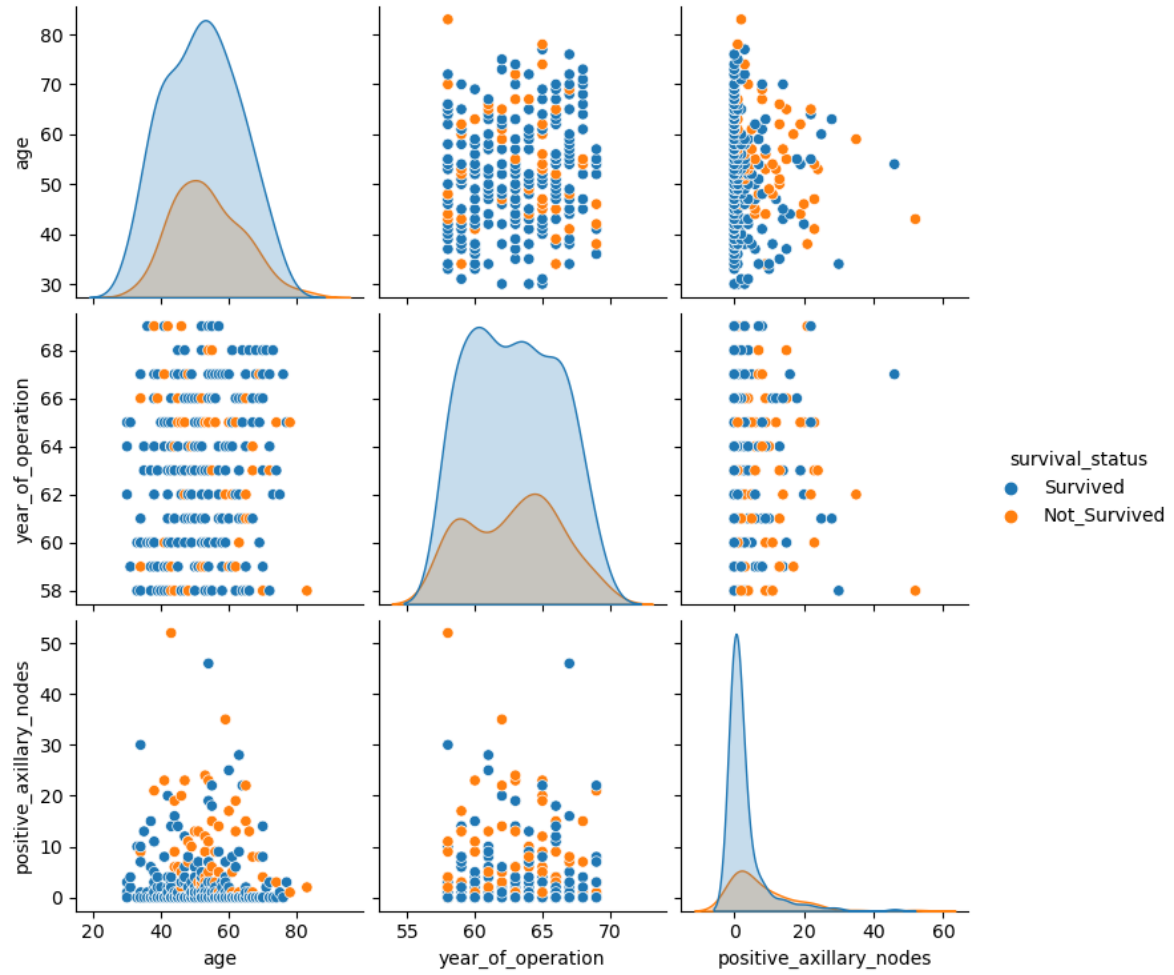
	age	year_of_operation	positive_axillary_nodes
count	306.000000	306.000000	306.000000
mean	52.457516	62.852941	4.026144
std	10.803452	3.249405	7.189654
min	30.000000	58.000000	0.000000
25%	44.000000	60.000000	0.000000
50%	52.000000	63.000000	1.000000
75%	60.750000	65.750000	4.000000
max	83.000000	69.000000	52.000000

I then transformed the numerical 'survival_status' column to categorical for better interpretability. I mapped 1 to 'Survived' and 2 to 'Not_Survived'.

After preprocessing, I started visualizing the dataset. I used Seaborn to plot the patient's age against the number of positive axillary nodes, color-coding the points by the survival status. This gave me a visual insight into the potential relationships between these attributes.



To further understand the pairwise relationships among the attributes, I utilized Seaborn's pairplot function. The plots again had color-coding based on the survival status:

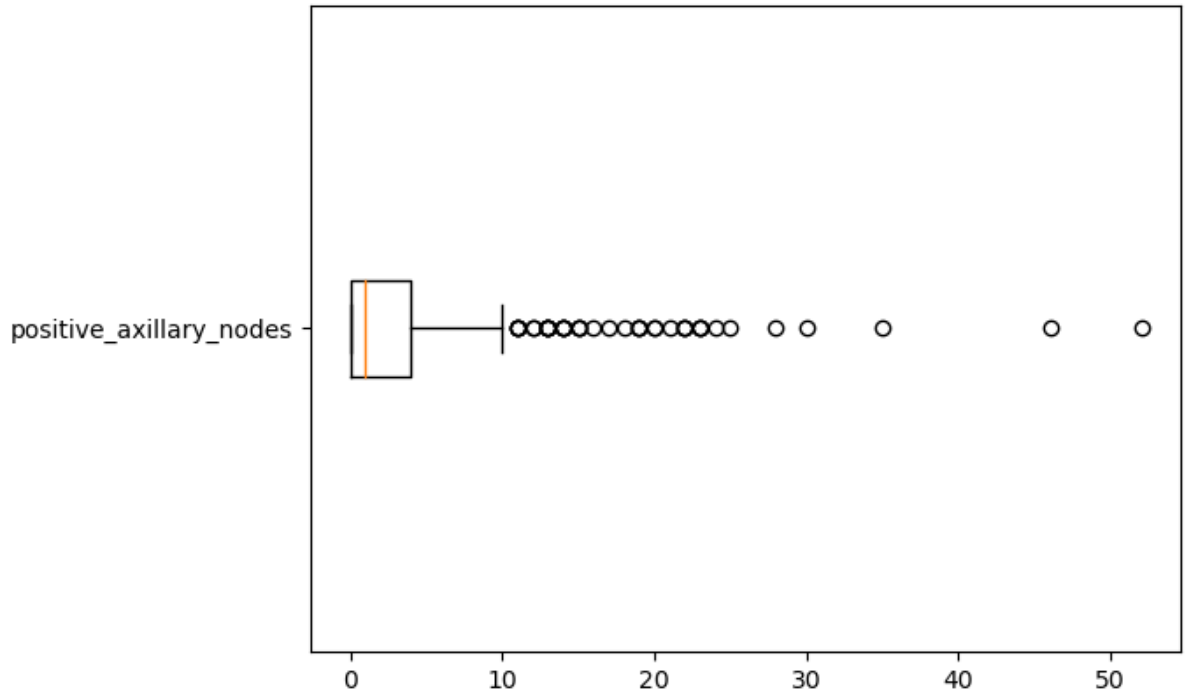


By the end of this step, I had a solid understanding of the structure and relationships within my data. This will be crucial as I move on to the next steps of data preprocessing and model training.

Step 2:

In this step of my assignment, I focused on improving data quality by identifying and handling outliers and missing values.

Initially, I visually inspected the dataset for outliers using a boxplot for the 'positive_axillary_nodes' attribute, which revealed some extreme values. To systematically detect and remove these outliers, I calculated the Interquartile Range (IQR) and defined an outlier condition based on it. Outliers were then removed from the dataset, enhancing its reliability.



Next, I checked for missing values, a critical step as these can lead to inaccuracies in the model. I found no missing values, but if there had been any, my plan was to fill them with the mean value of their respective column, a common strategy for handling missing data.

Finally, I compared the shape of the cleaned dataset with the original one. The reduction in the number of rows confirmed the successful removal of outliers. This rigorous data cleanup has prepared my dataset for the next steps of the project.

Step 3:

During the third stage of my project, I carried out essential data pre-processing tasks to ensure that the dataset is well-suited for the upcoming model training.

Firstly, I divided the data into features and the target. The features consist of all columns except the 'survival_status' which was treated as the target. Subsequently, I converted the features to float type, ensuring that all data could be processed uniformly.

The target column contained string values, which are not suitable for machine learning algorithms. Therefore, I applied Label Encoding to transform these string labels into numeric values (0/1). This makes the data compatible with the mathematical operations used in machine learning algorithms.

Next, I split the dataset into training and testing sets, following the recommended ratio of 70%-30%. This ensures that the model will have sufficient data for learning while leaving a significant amount for performance evaluation.

Lastly, I employed the MinMaxScaler to normalize the data. Normalization scales the values in the dataset to a specific range (between 0 and 1 in this case), improving the performance of the learning algorithms by ensuring all features have the same scale. The scaler was fit only on the training data to prevent information leakage from the test set, a critical practice to ensure valid model evaluation.

Step 4:

In Step 4 of my project, I designed and trained a neural network for the Haberman's Survival dataset classification task. Using Keras, I created a sequential model with an input layer of 10 neurons which uses the relu activation function and a sigmoid-activated output layer. I set the learning rate at 0.3 and a maximum of 3000 training epochs.

I compiled the model with an Adam optimizer and binary cross-entropy loss function. To automatically stop training when the loss failed to improve beyond 1e-5, I used the EarlyStopping callback in Keras. Additionally, I utilized a CustomCallback function to display model metrics every 100 epochs, as required.

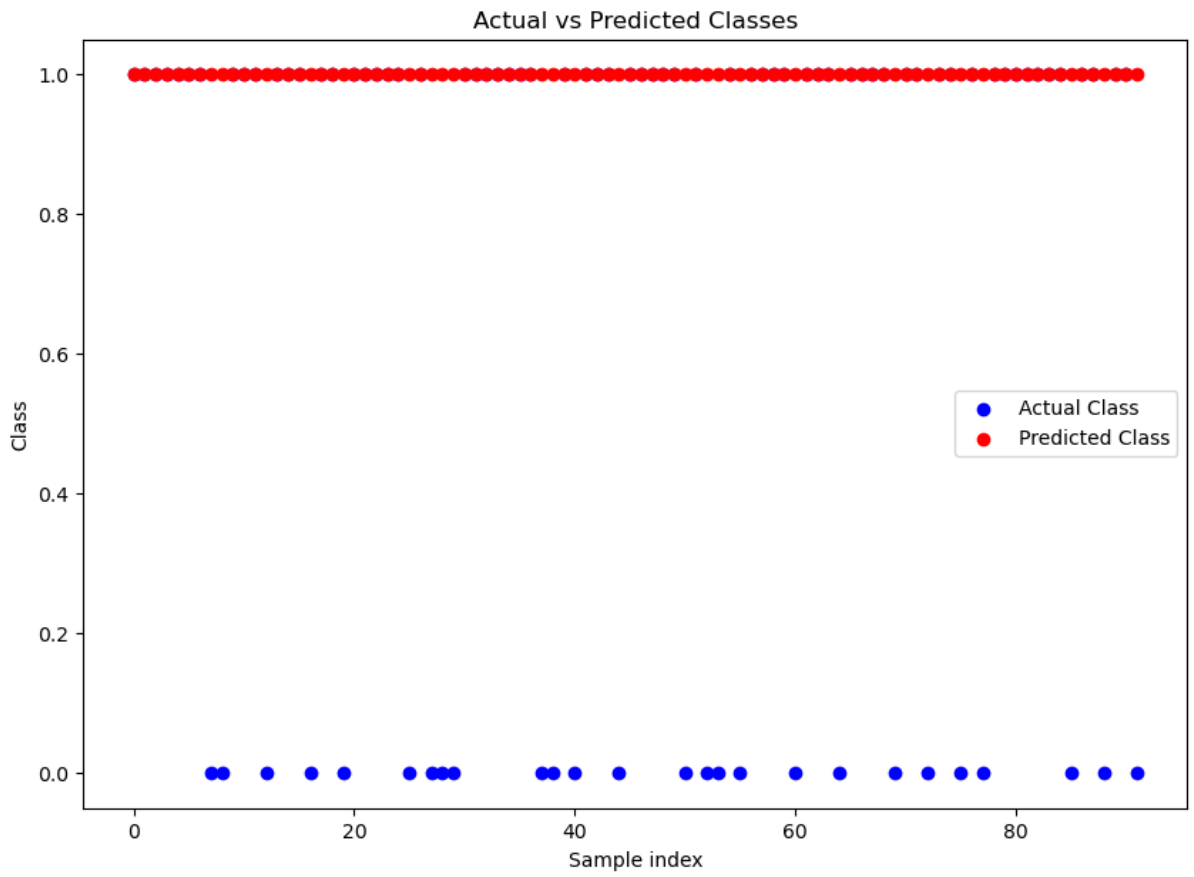
However, because of the accuracy paradox, which I will explain in the next step, the training stopped before reaching 100 epochs due to no significant improvement in the loss. Despite the early halt, the model's construction and training served as a valuable learning experience in optimizing neural network performance.

Step 5:

In Step 5, I tested the trained neural network on unseen data and evaluated its performance. Initially, the model was presented with the test set, and the accuracy was calculated. The model achieved an accuracy of approximately 72%, which may initially appear to be satisfactory.

However, upon closer inspection of the predicted classes and their comparison to the actual classes, an issue related to the Accuracy Paradox became evident. The Accuracy Paradox is a phenomenon where a model's accuracy can be misleadingly high, especially in cases where the dataset is imbalanced. In the Haberman's Survival dataset, the majority class is 'Survived'. It appeared that the model predominantly predicted class 1 ('Survived'), which contributed to the high accuracy score.

The classified patterns were plotted using the matplotlib library, where the comparison between actual and predicted classes further emphasized this disparity. The scatter plot clearly showed the model's tendency to predict almost all instances as 'Survived', resulting in poor performance for 'Not_Survived' instances.



This paradox illuminates an important limitation in relying solely on accuracy as a performance metric in classification problems, especially with imbalanced datasets. It indicates the need to potentially tweak the model parameters or use alternate strategies, such as oversampling the minority class or using a different performance metric that is more sensitive to false negatives and false positives.

```
3/3 [==============================] - 0s 1ms/step - loss: 0.5955 - accuracy: 0.7174
3/3 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        26
           1       0.72      1.00      0.84        66

    accuracy                           0.72        92
   macro avg       0.36      0.50      0.42        92
weighted avg       0.51      0.72      0.60        92
```

While the model showed reasonable accuracy, it was clear that improvements are needed for it to be a reliable predictor of survival. The next steps would involve further optimization of model parameters and potentially adopting different strategies to deal with class imbalance.

Step 6:

In this phase of the project, I turned my focus towards optimizing my neural network model. The intent behind this step was to enhance the model's predictive performance by tuning various hyperparameters. I chose four key parameters for tuning: the number of neurons in the hidden layer, the batch size, the number of epochs, and the learning rate.

In my base model, I initially employed 10 neurons in the hidden layer with a batch size of 16, a learning rate of 0.3, and set it to train over 3000 epochs. The activation function for the hidden layer was 'relu', while the output layer used 'sigmoid' for its binary classification task. I used the Adam optimizer with its default parameters, setting our chosen learning rate.

The next step in the process was to perform a Grid Search Cross-Validation, a powerful hyperparameter tuning technique. Grid search systematically works through multiple combinations of hyperparameter tunes, cross-validating as it goes to determine which tune gives the best performance.

I set a grid of parameters for this technique to explore. For the neurons in the hidden layer, I used values of 5, 10, 15, and 20. The batch sizes tested were 8, 16, 32, and 64. I experimented with the model training for 100, 150, 200, and 250 epochs. Also, I set learning rates of 0.001, 0.01, 0.1, and 0.3.

Once I ran the Grid Search with the specified parameters and my dataset, it returned the best parameters for my model: 5 neurons, a batch size of 32, 200 training epochs, and a learning rate of 0.1.

Following this, I compared the performance of my base model with the new optimized model by presenting the results in a table format. This comparison was instrumental in visualizing the effect of the changes I made to my model parameters.

Here are the first 10 configurations (including the base model in the first line) ordered by their performance:

	param_neurons	param_batch_size	param_epochs	param_learning_rate	mean_test_score	rank_test_score
0	10	16	3000	0.3	0.717391	0
169	5	32	200	0.1	0.780060	1
180	20	32	250	0.001	0.775235	2
211	15	64	150	0.001	0.770931	3
114	10	16	250	0.001	0.770866	4
8	20	8	100	0.01	0.770670	5
82	10	16	150	0.001	0.766236	6
38	10	8	200	0.01	0.766171	7
245	5	64	250	0.01	0.766171	7
181	5	32	250	0.01	0.766171	7

In summary, hyperparameter tuning was a crucial step in my project. It allowed me to optimize my model by systematically searching through a multitude of potential configurations to identify the one that offered the best performance. This approach resulted in a model with improved predictive capabilities, a noteworthy achievement in my project's progression.

Step 7:

In this stage of the project, my focus has shifted towards studying the impact of different data split proportions on the performance of the optimized neural network model. The main goal is to understand how the distribution of training and testing data affects the model's accuracy.

To conduct the experiment, I used the best model architecture identified in the previous step. It consists of one hidden layer with 5 neurons and utilizes the Adam optimizer with a learning rate of 0.1. The model was trained for 200 epochs with a batch size of 32.

I aimed to observe the model's performance across various dataset split rates. I considered four split rates: 50%-50%, 60%-40%, 80%-20%, and 90%-10%. The first value represents the proportion of training data, while the second value indicates the proportion of test data.

For each split ratio, I divided the dataset into training and testing sets. Then, I trained the model using the training set and evaluated its performance on the corresponding testing set. I recorded the accuracy score for each iteration.

To make it easier to compare the results, I consolidated the accuracy scores into a table. This table shows the split ratios alongside their respective accuracy scores.

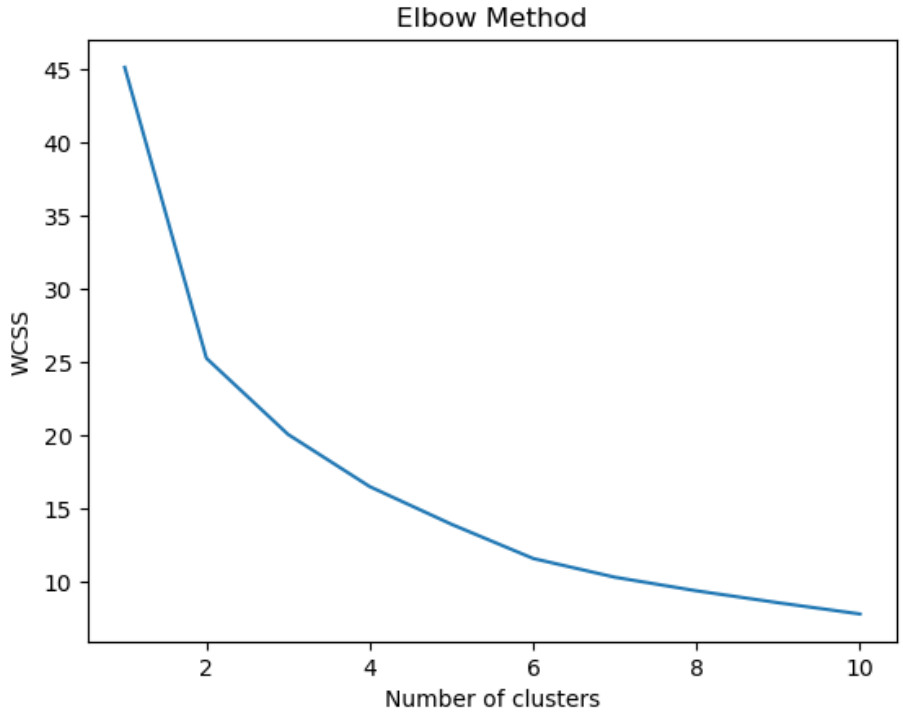
	split_ratio	test_accuracy
0	0.5	0.764706
1	0.6	0.739130
2	0.8	0.734694
3	0.9	0.713768

Through this experiment, I noticed that the model's performance varied depending on the data split ratio. This can be attributed to the different amounts and distributions of training and testing data that the model encountered. It became clear that finding the right balance in data distribution is crucial for the effectiveness of a machine learning model.

Step 8:

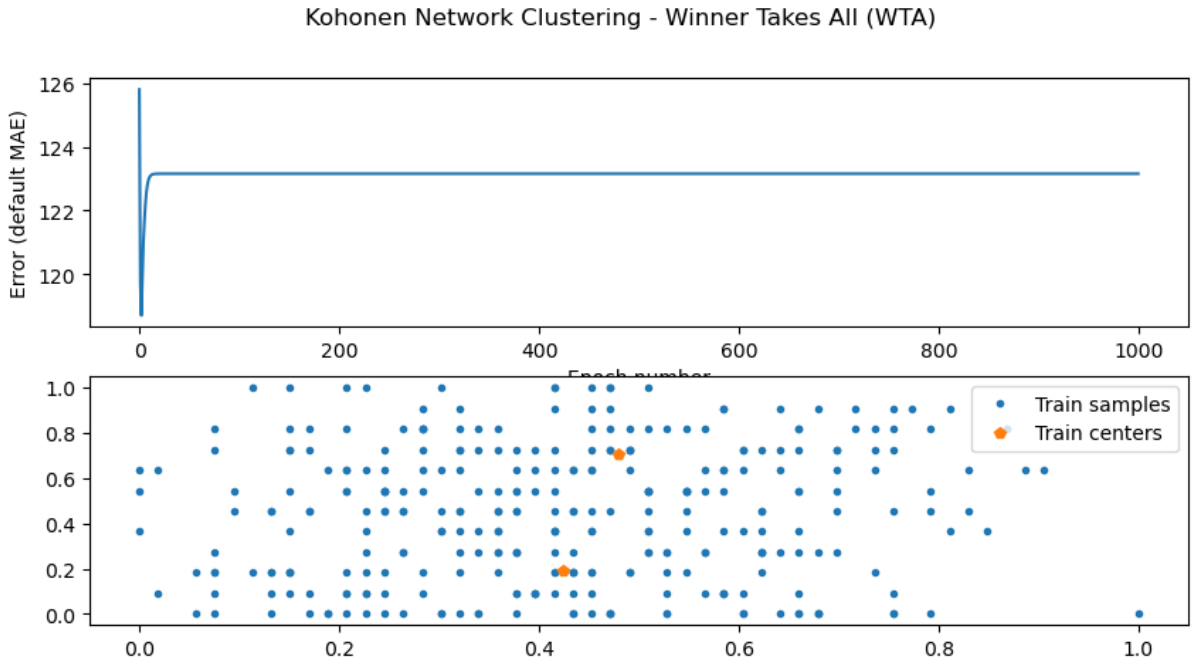
For the eighth step in this project, I moved from supervised learning into unsupervised learning. In this instance, the aim was to cluster data, which involved removing the column containing class labels.

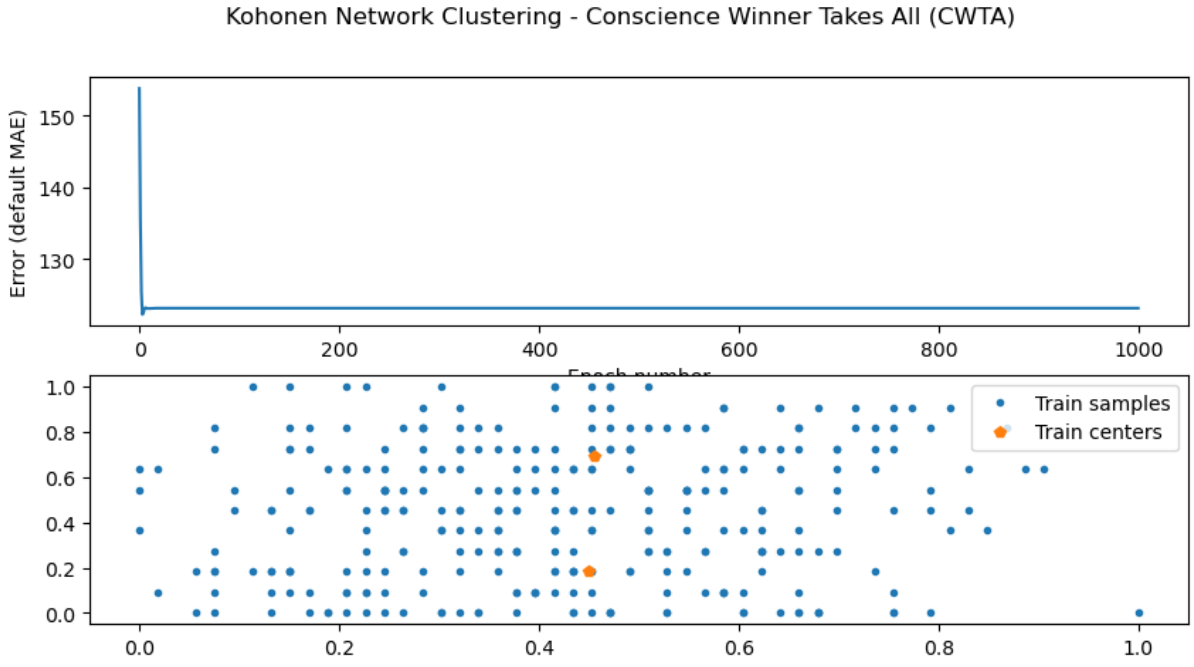
I first normalized the data, ensuring all features had equal contributions. I then employed the Elbow method and the Silhouette score to determine the optimal number of clusters.



Upon using these methods, an inconsistency was observed: the Elbow method suggested two clusters, which is consistent with my knowledge of the data. In contrast, the Silhouette score indicated four as the optimal number of clusters. This observation underscored the importance of considering multiple methods and interpreting the results with an understanding of the dataset.

Subsequently, I utilized the Neurolab library to implement the Kohonen network. I adopted the Winner Takes All (WTA) and Conscience Winner Takes All (CWTA) training methods for a total of 1000 epochs. By examining the error progression and final clusters produced, I could assess the effectiveness of the Kohonen network's unsupervised learning capabilities.





In conclusion, this step highlighted a key observation: while algorithms like the Elbow method and Silhouette score provide valuable insights, they are not infallible. They gave different recommendations for the optimal number of clusters, reinforcing the need for informed decision-making in machine learning. It was also exciting to see how well the Kohonen network performed in terms of clustering the data. Both the WTA and CWTA training approaches offered valuable insights and further solidified my understanding of unsupervised learning.

Step 9:

In this course I learned the basics of Artificial Intelligence and Neural Networks. I learned how a Neural Network is constructed, how it works, what is behind the “artificial” brains. Being my first course on this topic, I feel like it was a valuable experience and I wish to continue on this path, maybe someday constructing my own Network :). I would say the most important step to follow when implementing a neural network is finding the right architecture and that is obtained only by experimenting with several values for the parameters of the network. I learned that only through trial and error one develops a skill, and mastering Artificial Intelligence is a long way to go, but this course put me on the right path.