Explainability

How I approached this:

1. This task was very straightforward when we talk about the model's implementation here.
2. First was understating the data, so after importing the files, I tried to understand how the data looke d like, if there was any empty field or not.
    a. The data looked clean and there was not any of the field which had any of the null values in t he train data.
3. Then after importing the files, I then applied the following functions.

def evaluate_model(model, x_train, y_train, x_test, y_test, label):

1. The method in invoked within each of the major functions and has only two functions, one is to fit wh atever model is passed as an argument with the train data and then predict the result. The second wo rk of the function is to display/print the accuracy, precision, recall and F1 score of the model on the t est data after training.
2. These were the same steps that were used in all the major functions, so to reduce the length of the co de and increase reusability of the code, I used this function.

def DT_explain(train_file,test_file):

1. After segregating the descriptive features and target feature, I went ahead with training the data first with the default parameters and with normalization, without normalization and with standard scalar.
2. I also used the model in the same function to print not only accuracy but precision, recall and f1 score for both the train and test data and see how the results were.
3. I then used the plot_tree function to show the data directly but the result for the image had text over written in it and it was difficult to read.
4. So, instead of using only plot_tree alone, I used the function with the mathplot lib to plot the image an d after some tweaking got a good result.
5. This function was straightforward, and I didn't encounter any issue other than finding the correct par ameters for the image to be clear and crisp.

def LR_explain(train_file,test_file):

1. I had worked with this a lot, so understanding this was not an issue.
2. I followed the same principle of the segregating the descriptive features and target feature described above.
3. I chose to go ahead with the default parameters but got error which was because the number of maxi mun_iterations were low. Now because of this, I chose the max_iterations as 1000 and it was working perfectly.
4. In this function the process involved same for fitting, the model, and predicting the result as dt_explai n. The only challenge was changing the classifier and printing the weights of the feature which helped me with the model explainability. Which we'll discuss later.

def MLP_explain(train_file, test_file):

1. This one seemed a bit hard to implement at first because the concept of **LOCO (Leave-One-Covaria te-Out)** was new to me.
2. Once I understood what this meant and devised a plan as to how I could approach it, it became e asier to me.
3. Reading the data was like the above functions. I then used the standard scalar rather than the no rmalize to normalize the data with the mean of the columns(feature) with a variation of 1. With t his the data was normalized and I then later used the same transformation applied on the traini ng data onto the test data to form the consistent data across train and test data.
4. I called the evaluate_model function to print the model's accuracy, f1 score, precision, recall and f1 score both on test and train data.

5. I then used the LOCO methodology and implemented it by training different instances of MLP m odel by eliminating one feature in different instances to find the most important feature.
6. I then used the evaluate_method function to print each of the model's accuracy, precision, recall and F1 score.

Metrics Used for the calculations and what they mean:

- **Accuracy:** Represents the proportion of correctly classified instances, offering a straightforward measure of the model's overall performance.
- **F1-Score:** Provides a balance between precision and recall, making it especially useful for imbalanced datasets by minimizing both false positives and false negatives.
- **Precision:** Measures the proportion of correctly predicted positive instances among all predicted positives, reflecting the reliability of positive classifications.
- **Recall:** Evaluates the proportion of actual positive instances correctly identified by the model, emphasizing its ability to capture all relevant cases.

Detailed Analysis and Results:

Logistic Regression:

Figure 1.

Logistic Regression Model and Variation	**Accuracy**	**Precision**	**Recall**	**F1 Score**
LR with Basic Normalization across rows (Test Data)	0.789	0.818	0.767	0.792
LR without Normalization (Test Data)	**0.970**	**0.967**	0.975	**0.9715**
LR with Standard Scalar (Test Data)	**0.970**	**0.967**	0.975	**0.9715**
LR with Basic Normalization across rows (Train Data)	0.816	0.833	0.794	0.813
LR without Normalization (Train Data)	0.965	0.958	0.972	0.965
LR with Standard Scalar (Train Data)	0.966	0.956	0.978	0.967

Figure 2.

Feature	LR with Standard Scaler	LR without Normalization	LR with Basic Normalization across rows
Salary	0.9003	0.0000	0.9820
Age	0.0802	0.0025	0.0350
Credit Score	7.1995	0.0858	1.1641
Debt	-1.7650	0.0001	7.4074

For Test Data (Refer Figure 1 and 2):

The results for the Logistic Regression model are straight forward. The way I tested for the Logistic Regression model was with three variations, one by using normalize function, which normalizes the data along the row. This is good, but with this I believed that the result for the model won't be good as the row data for each example would be different and this won't be a great way for the model to be trained on, and not to mention this will be different for the test data as well. This assumption was proved correct when I saw the accuracy, precision, recall and F1 score for this was very low. Then, I tried without normalization and saw that the model performed amazingly well, when comparing this with the normalized data. I then chose to use Standard Scalar, which proved to be better than the normalized data, as the data was normalized across the mean of the train data and by a variation of 1. The same transformation was applied to the test data as well. This resulted in the consistent normalization across all the train data and test data. The result for this though was exactly equal to the model trained and tested on non-normalized data.

Talking about the explainability, I can see why the model for normalization for rows (basic normalized function) failed to produce good result. The weights it considered for salary and age was like the weights of normalization using the standard scalar (best performing normalization), but it failed in giving the higher weight to the credit score and then failed miserably when it gave very high positive value to the debt. This led to debt being the most important feature for its prediction which led to it producing bad results. When talking about the LR model where I didn't use any type of normalization, the weights was given very low to the debt, but the credit score was given a very high value which would've led to great result, but one thing to notice is it gave 0 value to the salary. **Based on this we can infer that the credit score is the column which holds the most importance in predicting the approval or denial of the loan.**

For Train Data (Refer Figure 1 and Figure 2):

The result for train data, is though not advised to generate the inference for the model, as it will produce good result on the data that it has already seen before. This is proved by the model trained on the basic normalization; the model showed jump across all the metrics. This however cannot be said about the model that was not normalized or normalized using standard scalar. With this the model showed similar but low accuracy but had a bit better recall. Now as the model was able to generalize well which can be seen with the result from the test data it did so on the train data as well. But like I said, the metrics from the train data makes no sense when evaluating the model.

MLP Classifier:

Results:

Model Variant	Train/Test	Accuracy	F1 Score	Precision	Recall

Base MLP Model	Train	**0.9720**	**0.9723**	0.9666	**0.9781**
Base MLP Model	Test	0.9700	0.9714	**0.9696**	0.9733
Feature Removed: Salary	Train	0.9645	0.9649	0.9597	0.9702
Feature Removed: Salary	Test	0.9640	0.9657	0.9639	0.9676
Feature Removed: Age	Train	0.9705	0.9708	0.9665	0.9751
Feature Removed: Age	Test	0.9710	0.9724	0.9714	0.9733
Feature Removed: Credit Score	Train	**0.8510**	**0.8380**	**0.9245**	**0.7664**
Feature Removed: Credit Score	Test	**0.8210**	**0.8086**	**0.9197**	**0.7214**
Feature Removed: Debt	Train	0.9510	0.9513	0.9504	0.9523
Feature Removed: Debt	Test	0.9400	0.9429	0.9411	0.9447

For Test Data:
The MLP model was a bit tricky to explain compared to the other models. Since neural networks don't have direct feature weights like Logistic Regression, I had to use **LOCO (Leave-One-Covariate-Out)** to check feature importance. From the results, **credit score was clearly the most important feature** removing it caused the biggest drop in accuracy. This makes sense since credit score is a strong indicator of loan approval. Refer the table above for different result, we can see that the lowest accuracy was when credit score was left out which was at 82% with test data which is very low when we look at what the model got when we had credit other features dropped out.

For test data, the model performed well, and generalization was good. Removing **salary or age** didn't impact performance much, which means the model wasn't relying heavily on them. **Debt also played a role**, but not as much as credit score. One challenge with MLP is **explainability in deeper layers** is that, more complex the network, the harder it is to pinpoint exact feature influence. While feature importance was clear from LOCO, understanding how neurons interact in hidden layers would need a more detailed approach.

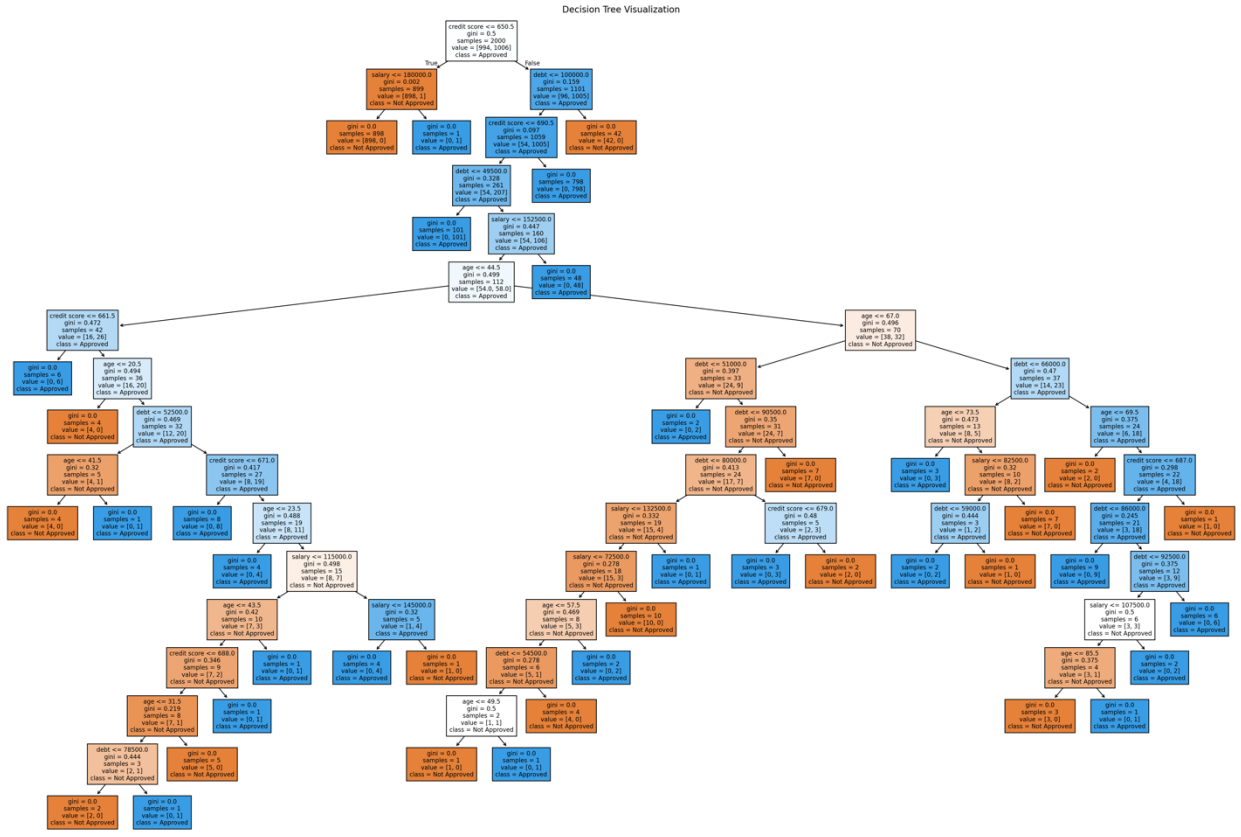
For Train Data:

Train results followed the same trend as test data. The model performed well, but since it had already seen this data, the results were naturally better. The key takeaway is that **credit score is the most important feature here**, and without it, the model struggles. While neural networks are powerful, their decision-making process isn't as transparent as decision trees or Logistic regression models, which is why the explainability can become a task.

Decision Tree:

Results:

Metric	**Test Data**	**Train Data**
Accuracy	0.970	1.0
F1 Score	0.971	1.0
Precision	0.966	1.0
Recall	0.977	1.0



The explainability comes easiest with the Decision Tree out of all the models in this. With decision Tree we can visualize the path it would've too to make the decision it made (as can be seen from the image above). The division of the nodes into true of false based on a condition can help us guide us on the direction to decide. Here the different features are divided into forming the decision tree, that later splits into the two nodes into

a left node and right node based on some condition. **But the feature which causes the first split provided the highest information gain and is the reason why it's chosen as the root node. In this model, the most important feature is the credit score.**

One thing to note that out of all the three model only decision tree was the one which got 1.0 result across all the metrics with the train data. This was because the decision tree grew whole and saturated all the condition based on the train data and thus gave us metrics of 1. This means that model learned the training data and overfitted to it. Though it was able to generalize well with the test data , but according to me, if the data would've been bigger, it would've costed us a lot of computation and having rigid answers due to overfitting of the data.

All in all, explainability was greatest with the decision tree with the visualization and then understanding how the model made the decision it made.

Conclusion:

Looking at the results from all three models, there's a clear difference in both performance and explainability. Logistic Regression worked best when using standard scaling or no normalization at all, showing that credit score was the key feature in predicting loan approvals. MLP was trickier to explain, but LOCO helped in understanding that credit score had the biggest impact, while salary and age didn't contribute much. Decision Tree was by far the easiest to explain since it visually showed how decisions were made, with credit score as the root node. But the problem was it memorized training data, which led to overfitting, which can be seen with how it produced 100% in accuracy, precision, recall and f1 score. In terms of explainability, Decision Tree was the best, then Logistic Regression, while MLP was the one which I can say is harder to interpret when we move deep in the layers. Across all models, credit score was the most important feature, making it clear that it plays the biggest role in loan approvals.

Challenges Faced:

1. While there were no challenges faced as such, there was some complications faced when implementing the code. To make the code more reusable I had to create the function evaluate_model which I had to change a couple of time to increase reusability in all the cases like when I thought to implement label to print the result for the removed feature in the MLP model so that I can reuse the code there as well.

2. How to implement the LOCO was all though simple, took me some time to think on how to implement this in a good way so that it's understandable and produces a good result. I was going on in wrong direction, in which I was trying to explain the explainability of the model from the weights and find the most important model. I thought the approach would work but then it didn't.

3. When trying to display the image of the decision tree directly from the plot_tree function, I could see that overlapping between the text which made it very hard to read. So, I had to use the math plot library to plot the tree so that it doesn't overlap. This involved some customizations of the parameters to get just the right image.