Machine Learning Assignment 5 Report

Luthfi Fuad | 6758090256

1 | Executive Summary

This report documents the end-to-end machine learning pipeline developed for Assignment 5 of Machine Learning, which addresses a binary classification problem: predicting whether a customer will purchase a travel product (ProdTaken) given demographic, economic, and behavioural data.The dataset, sourced from tourism_data.csv, comprises 3,209 records across 19 features. Before developing the model, the data underwent cleaning to resolve three irregularities: a misspelled gender category, redundant marital status labels, and two statistically unreliable "Free Lancer" entries in the Occupation column. The cleaned dataset was then split 90/10 into training and test sets, with the larger training proportion justified by the dataset's modest size.

Feature engineering was conducted across three fronts: outlier capping at the 99th percentile for high-variance numerical columns, interaction feature creation guided by Pearson correlation analysis, and ordinal tier mappings for designation and product categories to preserve their natural hierarchy.

The core model is a deep stacked ensemble combining Random Forest, Extra Trees, Gradient Boosting, Histogram Gradient Boosting, and a custom Keras MLP, fused by a Random Forest meta-learner through 5-fold cross-validated stacking. The MLP employs focal loss to address class imbalance, a funnel architecture (512→256→128), batch normalisation, and early stopping.

Two model variants were evaluated. The first, trained with ranking and categorical handling, achieved 94.08% accuracy and a weighted F1 of 0.9419. The second, trained without those features, outperformed it at 95.95% accuracy and a weighted F1 of 0.9594. Training curves for both models confirmed the absence of overfitting, with training and validation metrics remaining closely aligned throughout.

2 | Data Preparation & Analysis

For this assignment, I downloaded **tourism_data.csv**, from Class 5 Image Data under Supplementary Materials in myCourseVille.

tourism_data.csv consists of 3,209 rows and 19 columns. Rows correspond to different people being pitched to, while columns correspond to different categories of demographic, economic, and pitch-related data, as well as a column that explains whether the purchase was made or not. The columns can be categorized further into ordinal, categorical and numerical data as such:

Column Name	Data Type	Data Category	Range or Potential Values
Age	Integer	Numerical	18-60
TypeofContact	String	Nominal	Self Enquiry, Company Invited
CityTier	Integer	Ordinal	1 - 4

DurationOfPitch	Integer	Numerical	5 - 36
Occupation	String	Nominal	Salaried, Free Lancer, Small Business, Large Business
Gender	String	Nominal	Male, Female
NumberOfPersonVisiting	Integer	Numerical	1 - 4
NumberOfFollowups	Integer	Numerical	1 - 6
ProductPitched	String	Ordinal	Basic, Standard, Deluxe, Super Deluxe, King
PreferredPropertyStar	Integer	Ordinal	3 - 5
MaritalStatus	String	Nominal	Single, Unmarried, Married, Divorced
NumberOfTrips	Integer	Numerical	1 - 20
Passport	Integer	Nominal	0 - 1
PitchSatisfactionScore	Integer	Numerical	1 - 5
OwnCar	Integer	Nominal	0 - 1
NumberOfChildrenVisiting	Integer	Numerical	0 - 3
Designation	String	Ordinal	Manager, Senior Manager, AVP, VP, Executive
MonthlyIncome	Integer	Numerical	1000 - 34246
ProdTaken	Integer	Nominal	0 - 1

ProdTaken	0	1	% of Yes
TypeofContact			
Self Enquiry	1864	415	18.21%
Company Invited	725	204	21.96%
CityTier			
1	1735	339	16.35%
2	90	36	28.57%
3	764	244	24.21%

Occupation			
Large Business	217	87	28.62%
Small Business	1085	254	18.97%
Free Lancer	0	2	100.00%
Salaried	1287	276	17.66%
Gender			
Female	955	209	17.96%
Male	1534	389	20.23%
ProductPitched			
Basic	876	381	30.31%
Standard	480	89	15.64%
Deluxe	994	134	11.88%
Super Deluxe	168	10	5.62%
King	71	5	6.58%
PreferredPropertyStar			
3	1648	328	16.60%
4	477	122	20.37%
5	464	169	26.70%
MaritalStatus			
Divorced	558	77	12.13%
Single	316	194	38.04%
Married	1314	221	14.40%
Passport			
0	1984	277	12.25%
1	605	342	36.11%
OwnCar			
0	1011	246	19.57%
1	1578	373	19.12%
Designation			
Executive	876	381	30.31%
VP	71	5	6.58%

AVP	168	10	5.62%
Senior Manager	480	89	15.64%
Manager	994	134	11.88%
ProdTaken			
0	2589	0	0.00%
1	0	619	100.00%

For every non-numerical category (ordinal, categorical), I counted each entries' quantity of 0s and 1s in its corresponding ProdTaken column. Through this, I identified broad patterns and outliers.

Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiting	NumberOfFollow	ProductPitched	PreferredProperty	MaritalStatus
30	Company Invited	1	21	Salaried	Fe Male	3	4	Standard	3	Unmarried
29	Company Invited	1	7	Small Business	Male	3	5	Basic	4	Married
28	Self Enquiry	1	24	Salaried	Female	3	4	Basic	3	Single
25	Self Enquiry	1	9	Salaried	Male	2	3	Basic	5	Married
21	Company Invited	1	13	Salaried	Female	4	5	Basic	3	Unmarried
30	Self Enquiry	1	7	Large Business	Male	3	4	Basic	3	Single
36	Self Enquiry	3	9	Salaried	Male	2	4	Standard	3	Married
38	Self Enquiry	2	13	Salaried	Male	4	4	Basic	5	Married
34	Self Enquiry	3	16	Small Business	Fe Male	4	4	Deluxe	5	Unmarried
41	Self Enquiry	3	23	Small Business	Male	4	4	Standard	3	Married
37	Self Enquiry	1	8	Free Lancer	Male	3	4	Basic	3	Single
24	Self Enquiry	1	15	Small Business	Male	4	5	Basic	3	Unmarried
41	Company Invited	1	11	Salaried	Male	3	4	Basic	5	Married
24	Self Enquiry	1	23	Salaried	Female	3	3	Basic	4	Divorced
33	Self Enquiry	1	9	Salaried	Male	4	4	Basic	4	Single
53	Self Enquiry	3	30	Salaried	Male	3	3	Standard	5	Unmarried
27	Company Invited	2	28	Salaried	Female	2	4	Basic	4	Married
60	Company Invited	1	21	Salaried	Female	3	4	Standard	3	Single
27	Self Enquiry	1	31	Large Business	Male	4	6	Basic	3	Single
40	Company Invited	3	30	Salaried	Fe Male	3	1	Super Deluxe	4	Unmarried

Irregularities in the Gender and Marital Status columns

D	E	F
DurationOfPitch	Occupation	Gender
9	Free Lancer	Male
8	Free Lancer	Male
7	Large Business	Male
31	Large Business	Male
9	Large Business	Male
13	Large Business	Male
14	Large Business	Female

Outliers in the Occupation column

I found that in the occupation category, there were only 2 data points with the "Free Lance" value, both of which corresponded to 1 on the ProdTaken column. I also noticed that some Gender entries were misspelled "Fe Male" instead of "Female". Additionally, "Single" and "Unmarried" in the Marital Status column referred to the same thing.

I wrote a python script to rectify all of these errors, **clean_data.py**. It makes those 3 changes:

- Drops rows that have an Occupation value of "Free Lancer"
- Changes "Fe Male" in the gender column to "Female"
- Replaces "Unmarried" to "Single" in the MaritalStatus column) and outputs a new file titled **tourism_data_cleaned.csv**

I wrote another python script (**split_data.py**) that makes a 90/10 split with **clean_data.py**, turning it into **test_data.csv** (10%) and **train_data.csv** (90%). I chose a 90/10 split over an 80/20 or an 85/15 split because the amount of data given is not that much, only 3,209.

3 | Feature Extraction

Features

3A | Outlier Capping

For numerical columns that have high variability (DurationOfPitch, NumberOfTrips, MonthlyIncome), I dropped data points that are above the 99th percentile to get rid of any outliers.

3B | Interaction Engineering

In order to find what new columns of data could be created by combining existing data, I looked into the correlations of every numerical variable with each other using the Pearson product-moment correlation coefficient (CORREL function in Google Sheets).

	Age	DurationOfPitch	NumberOfPersonVisiting	NumberOfFollowups	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome
Age	100.00%	-0.36%	-2.52%	-2.79%	16.09%	-4.62%	42.32%
DurationOfPitch		100.00%	7.33%	2.15%	-0.24%	3.87%	3.11%
NumberOfPersonVisiting			100.00%	32.19%	18.90%	59.16%	15.96%
NumberOfFollowups				100.00%	12.75%	26.39%	13.49%

NumberOfTrips					100.00%	17.37%	12.29%
NumberOfChildrenVisiting						100.00%	16.22%
MonthlyIncome							100.00%

The variable-pairs with the highest correlation are NumberOfChildrenVisiting with NumberOfPersonVisiting (59.16%); and MonthlyIncome with Age (42.32%). I decided to create three new columns based on these pairs:

➢ NumberOfAdultsVisiting = NumberOfPersonVisiting - NumberOfChildrenVisiting
➢ IncomeToAgeRatio = MonthlyIncome / Age
➢ IncomeSeniority = MonthlyIncome * Age

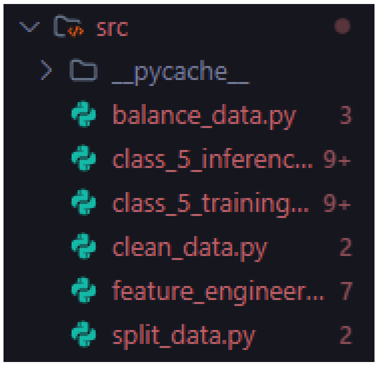
3C | Ranking & Categorical Handling

The ordinal categories (Designation, ProductPitched and DesignationTier) are treated by the code as just separate categories without an ordering, even though it would be useful to treat them like that. A string-based map was created that ranked the Designation, ProductPitched and DesignationTier like this:

Value	1	2	3	4	5
Designation and DesignationTier	Executive	Manager	Senior Manager	AVP	VP
ProductPitched	Basic	Deluxe	Standard	Super Deluxe	King

Implementation

Initially, I put the code that extracts all these features into my training and inference files (**class_5_training.py** and **class_5_inference.py**), but that means if I made any changes to it, I would have to make sure it is the same in both files. To avoid this problem, I refactored the feature extraction code to a new file titled **feature_engineering.py** which is referenced by both training and inference. I trained two different models: example**/luffy_class_5_training_model_v1.joblib** (which includes ranking & categorical handling) and **example/luffy_class_5_training_model_v2.joblib** (which doesn't). The second model performs better, as I will explain in the evaluation results.



4 | Building Model

Training Code 1

I used the code from class_4 and modified it to input **data/test_data.csv** instead of **data/bank.csv**

This script serves as my inference and evaluation pipeline for a previously trained binary classification model.

After training is complete, the model's predictive power means nothing unless I can verify it holds up against data it has never encountered. That is exactly what this script does — it loads my saved model, runs it against a held-out test set, and rigorously evaluates its performance.

First, all the training artefacts are loaded (model weights, the fitted StandardScaler, and the exact column list from training)

The model outputs a continuous probability for each customer. A a threshold of 0.47 rather than the conventional 0.5, which is a deliberate decision, because by lowering the threshold I am biasing the model toward catching more true positives at the expense of slightly more false positives, which is often the preferable trade-off in a sales conversion context where missing a genuine buyer is more costly than pursuing a non-buyer.

Configuration

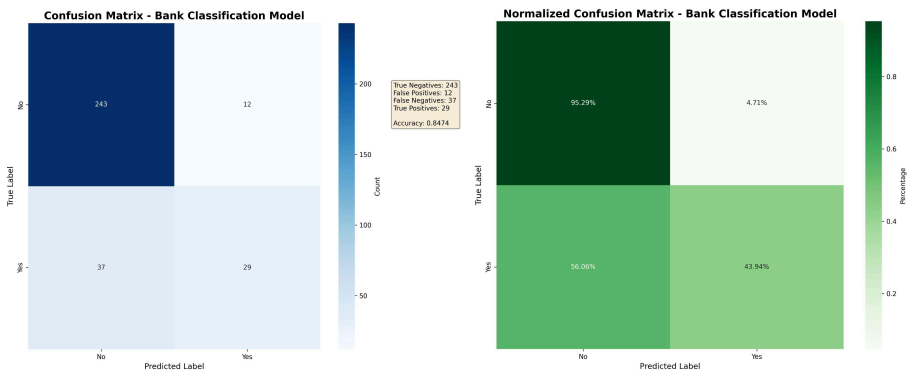
I set the test data path to **data/test_data.csv**, which was obtained through a python script that split the original csv file into 90% for training and 10% for testing. I set the model bundle path to **examples/luffy_class_5_training_model_v0.joblib**

Results

The results were not great. The model yielded an accuracy of 84.74%

```
============================================================
INFERENCE RESULTS
============================================================
Accuracy: 0.8474
AUC Score: 0.8824
```

```
Classification Report:
              precision    recall  f1-score   support

          No       0.87      0.95      0.91       255
         Yes       0.71      0.44      0.54        66

    accuracy                           0.85       321
   macro avg       0.79      0.70      0.73       321
weighted avg       0.83      0.85      0.83       321


Confusion Matrix:
[[243  12]
 [ 37  29]]
```



Confusion matrix & Normalized Confusion Matrix for luffy_class_5_training_model_v0.joblib

Training Code 2

Configuration

Same as the previous model, I set the training data to **data/train_data.csv**, the 90% split from the cleaned dataset. I defined the model bundle path to **output/luffy_class_5_training_model_v1.joblib**. For convenience, each section is labeled with comments in the training file (**class_5_training.py**) as well.

MLP Wrapper and Loss

I used focal loss because the dataset was imbalanced (2589 people didn't buy the product, 619 did). Focal loss solves this by making the model care more about the misclassified examples and less about the easy ones it's already getting right.

For the values, alpha=0.25 and gamma=2.0 I used are just default values in this use-case. I didn't really have a reason to change it since the problem is similar (not many positive cases, lots of negatives).

3 dense layers at 512 (ReLU) → 256 (ReLU) → 128 (ReLU) → 1 (sigmoid) with dropout rates of 0.3 and 0.4 for the first 2 layers.

For neural architecture, I went with 512 to 256 to 128 as a funnel. I start wide to give the network room to pick up a lot of different signals from the feature space, then gradually compress into more abstract representations like in an encoder. I didn't want to go wider than 512 because the dataset is only text (csv file)so the input dimensionality is in the hundreds at most, so 512 is already generous.

I put BatchNorm after every Dense layer because it lets me use a higher learning rate without things exploding, and it has a regularization effect because the normalisation statistics are noisy at the batch level. I didn't put it before the output layer because it would distort the final sigmoid probabilities.

I chose ReLU as the activation function since it's computationally efficient and works well. The decreasing rate was deliberate. The first layer has 512 neurons, which is a lot of capacity, so I applied stronger dropout (0.4) to force redundancy. By the time we're at the 256-layer, we're starting to compress so I reduced it to 0.3. After 128 I dropped it entirely because that layer feeds directly into the output and at that point I want a stable, coherent signal. The specific values of 0.3 and 0.4 came from experimenting (0.5 was too aggressive and the validation loss wouldn't come down, 0.2 wasn't regularising enough).

Adam = 0.001

0.001 is the default Adam learning rate.

epochs=100 and batch_size=64

100 epochs is really just an upper ceiling now that early stopping is in place. I don't expect to hit it. Batch size 64 is a middle ground: too small (like 16) and the gradient estimates are noisy and training is slow. too large (like 512) and the model generalises worse. 64 is a really common default for exactly this reason.

The patience=15 came from looking at the loss curves before I added early stopping. I noticed that whenever the model was genuinely improving, it would improve for at least a few more epochs. If 15 epochs go by with nothing meaningful happening, it's stalled. I set min_delta=0.001 because the loss scale with focal loss is smaller than regular cross-entropy, so tiny improvements below 0.001 are probably just noise rather than real learning.

5 | Evaluation results

Attempt 1: With Ranking & Categorical Handling

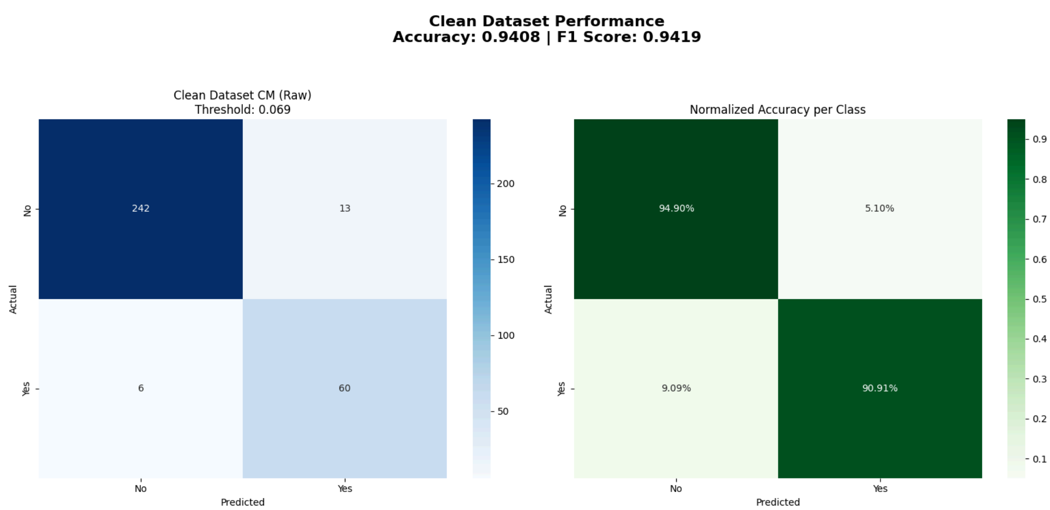
Here I ran inference on **examples/luffy_class_5_training_model_v1.joblib**, the model that I trained with ranking & categorical handling in feature extraction.

The training data took 37 epochs before stopping early.

```
Final Performance (Clean Dataset optimized):
------------------------------
Accuracy: 0.9408
Weighted F1: 0.9419
------------------------------
              precision    recall  f1-score   support

          No       0.98      0.95      0.96       255
         Yes       0.82      0.91      0.86        66

    accuracy                           0.94       321
   macro avg       0.90      0.93      0.91       321
weighted avg       0.94      0.94      0.94       321
```

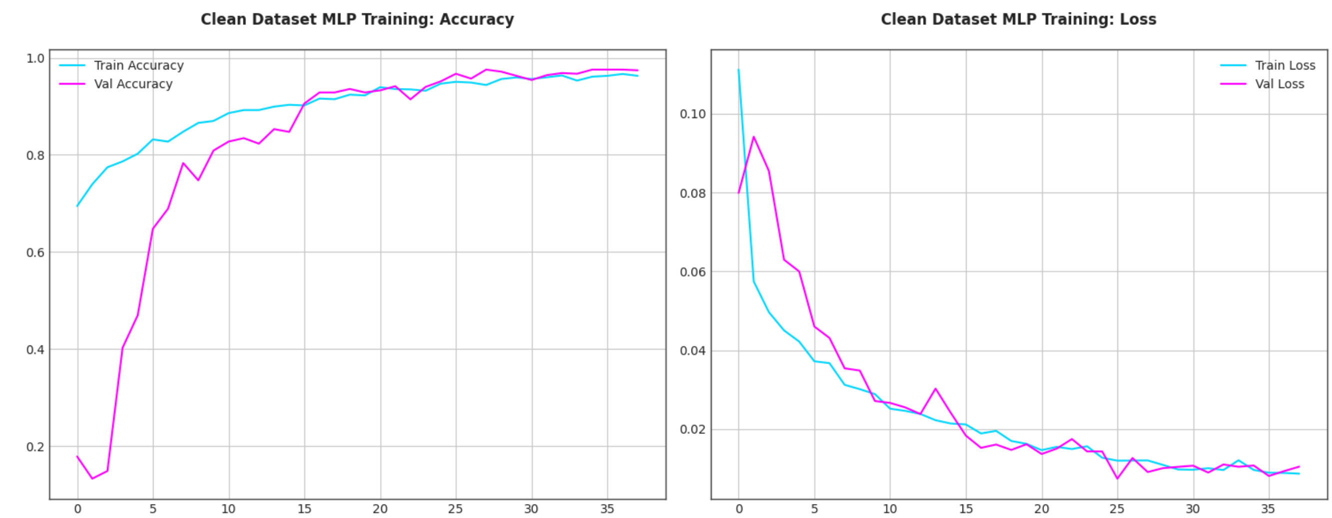

```
Final Performance (Clean Dataset optimized):
------------------------------
Accuracy: 0.9408
Weighted F1: 0.9419
------------------------------
              precision    recall  f1-score   support

          No       0.98      0.95      0.96       255
         Yes       0.82      0.91      0.86        66

    accuracy                           0.94       321
   macro avg       0.90      0.93      0.91       321
weighted avg       0.94      0.94      0.94       321
```


It returned the following values for precision, recall, f-1 score and support when I ran **class_5_inference.py** on **test_data.csv**. With an accuracy of 94.08%, this was the second best result I got. It still struggles a lot with the precision for Yes, having a value of 82%.



output/luffy_class5_v1_confusion_matrix.png

To ensure that these positive results aren't the result of overfitting, I monitored the training accuracy, validation accuracy, training loss and validation loss throughout every epoch. These values can be seen in the charts below. As the values (train accuracy to val accuracy & train loss to val loss) remain very close to each other, it can safely be said that the model is not overfitting.



output/luffy_class5_v1_training_history.png

Attempt 2: Without Ranking & Categorical Handling

I set the model bundle path to **examples/luffy_class_5_training_model_v0.joblib**

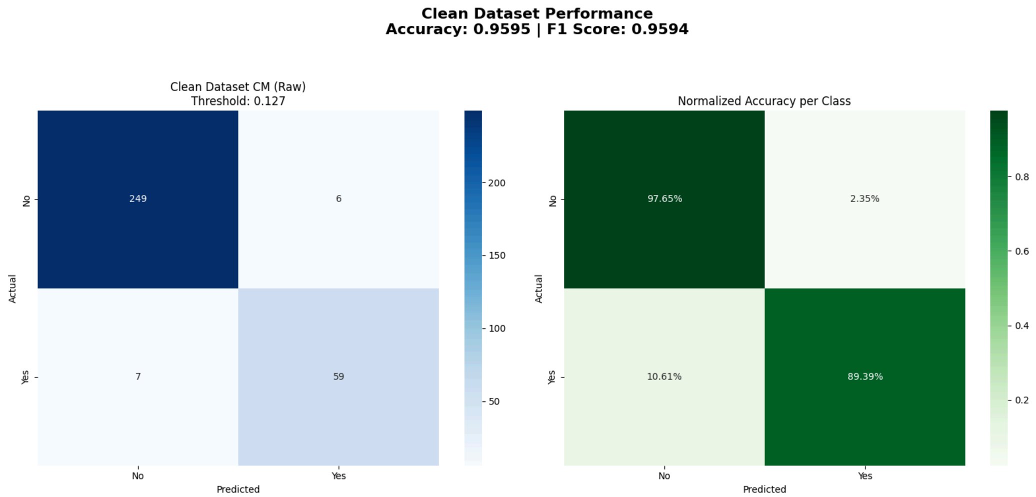
The training data took 62 epochs before stopping early.

It returned the following values for precision, recall, f-1 score and support when I ran **class_5_inference.py** on **test_data.csv**. With an accuracy of 95.95%, this was the best result I've gotten so far. It still struggles a bit in the recall section for Yes, having a value of 89% because the dataset ultimately has more information on the "No's".

```
Final Performance (Clean Dataset optimized):
------------------------------
Accuracy: 0.9595
Weighted F1: 0.9594
------------------------------
              precision    recall  f1-score   support

          No       0.97      0.98      0.97       255
         Yes       0.91      0.89      0.90        66

    accuracy                           0.96       321
   macro avg       0.94      0.94      0.94       321
weighted avg       0.96      0.96      0.96       321
```

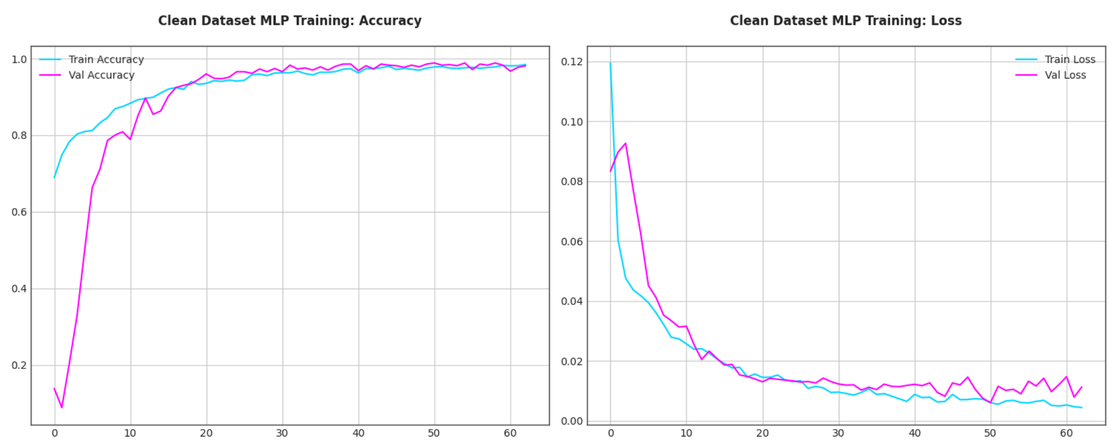

```
Final Performance (Clean Dataset optimized):
------------------------------
Accuracy: 0.9595
Weighted F1: 0.9594
------------------------------
              precision    recall  f1-score   support
```

	precision	recall	f1-score	support
No	0.97	0.98	0.97	255
Yes	0.91	0.89	0.90	66
accuracy			0.96	321
macro avg	0.94	0.94	0.94	321
weighted avg	0.96	0.96	0.96	321



output/luffy_class5_v2_confusion_matrix.png

Once again, to ensure that these positive results aren't the result of overfitting, I monitored the training accuracy, validation accuracy, training loss and validation loss throughout every epoch. These values can be seen in the charts below. As the values (train accuracy to val accuracy & train loss to val loss) remain very close to each other, it can safely be said that the model is not overfitting.



output/luffy_class5_v2_training_history.png

5 | Real Life Applications

At its core, this is a pipeline that ingests tabular data (rows of observations, columns of attributes) and learns to predict a binary outcome. This structure can be generalized. Any domain that collects structured records about entities and wants to predict something about them can adopt a pipeline like this with minimal architectural changes.

For example, in medicine, it could predict whether a patient is likely to develop a condition given their clinical measurements. In education, it could flag students at risk of dropping out based on attendance and assessment records. In infrastructure, it could predict whether a piece of equipment is likely to fail given sensor readings. All aspects generalize because the underlying mathematical problem is the same regardless of what the rows and columns actually represent.