Machine Learning Internship Report

Submitted by: Yatharth Kumar Saxena

Objective:

To build a classification model using Random Forest to predict the Target variable from customer data. The goal is to compare performance between raw and preprocessed datasets using two different train-test splits: **70:30** and **80:20**.

Step-by-Step Explanation

1. Importing Required Libraries

The project started by importing essential Python libraries:

- pandas – for data handling
- numpy – for numerical computations
- scikit-learn – for model building, data preprocessing, and evaluation

These libraries enabled efficient data loading, transformation, model training, and performance assessment.

2. Loading and Preparing the Dataset

The dataset was loaded from a CSV file named:
Yatharth Kumar Saxena - ml_preprocessing_dataset_1000.csv

A .copy() of the dataset was stored in two versions:

- df_raw → to retain the raw dataset structure (no imputations)
- df_clean → to apply full preprocessing and transformations

The columns were stripped of extra whitespace to ensure consistency.

3. Initial Transformation – Target Encoding

The Target column was found to be of type object. To allow model training:

- It was label-encoded in both df_raw and df_clean.

---

4. Raw vs Cleaned Dataset Processing

Raw Dataset (df_raw)

- No missing value handling
- All categorical (string) features except Legacy_Customer_ID and Target were **label encoded directly**

Cleaned Dataset (df_clean)

- Categorical missing values were filled with **mode**
- Numerical missing values were filled with **mean**

- After imputation, all object-type columns were label-encoded

5. **Feature-Target Separation**

   The dataset was split into:

- **X** → All columns except Target and Legacy_Customer_ID
- **y** → Only the Target column

   This separation was done uniformly for both raw and cleaned datasets.

6. **Model Evaluation Pipeline**

   A reusable function evaluate_model() was created to:

- Split data into **training**, **validation**, and **testing** sets
- Apply **StandardScaler** if preprocess=True
- Train a **Random Forest Classifier**
- Print:
  - Accuracy
  - Classification Report (Precision, Recall, F1)
  - Confusion Matrix

   Both 70:30 and 80:20 train-test splits were tested with and without preprocessing.

Results

Split: 70:30 | Preprocessed: (Raw Data)

- **Accuracy**: 59.13%
- **F1-Score (Class 0 / Class 1)**: 0.72 / 0.22
- **Macro Avg F1**: 0.47
- **Confusion Matrix**: [[161 27]
  [96 17]]

Split: 70:30 | Preprocessed: (Cleaned Data)

- **Accuracy**: 57.80%
- **F1-Score (Class 0 / Class 1)**: 0.72 / 0.17
- **Macro Avg F1**: ~0.45
- **Confusion Matrix**: [[110 18]
  [63 9]]

Split: 80:20 | Preprocessed: (Raw Data)

- **Accuracy**: 61.5%
- **F1-Score (Class 0 / Class 1)**: 0.75 / 0.21
- **Macro Avg F1**: 0.48
- **Confusion Matrix:** [[113 15]
  [62 10]]

Split: 80:20 | Preprocessed: (Cleaned Data)

- **Accuracy**: 61.5%
- **F1-Score (Class 0 / Class 1)**: 0.75 / 0.19
- **Macro Avg F1**: 0.47
- **Confusion Matrix**: [[114 14]
  [63 9]]

Observations

- The model consistently performed **better on Class 0**, indicating a possible **class imbalance**.
- Preprocessing improved consistency and structure, but did **not significantly improve accuracy or recall** for Class 1.
- Across both splits, **Class 1 recall** remained low (~14% or less), limiting overall F1 score and macro average.
- Standard scaling and missing value treatment did not change the outcome drastically due to Random Forest's tree-based nature.

Conclusion

This internship project provided hands-on experience in building an end-to-end machine learning pipeline:

- From **data ingestion** and **cleaning**
- To **model training**, **evaluation**, and **comparison** across datasets

Although the Random Forest Classifier achieved only ~61% accuracy, the project underlines the importance of:

- **Data quality**
- **Class balancing**
- **Model choice**
- **Hyperparameter tuning**

Future enhancements may include:

- SMOTE / undersampling for **balancing**
- Trying **XGBoost, SVM, or Logistic Regression**
- Applying **GridSearchCV** for hyperparameter optimization