CREDIT CARD BEHAVIOUR SCORE PREDICTION

Prachi Sawant
23112073

Abstract

This project aims to build a forward-looking behavioural score model to predict whether a credit card customer will default in the upcoming billing cycle. We applied advanced data preprocessing, financial feature engineering, and supervised classification models. The model is evaluated using F2-score, AUC-ROC, and threshold tuning strategies that align with real-world risk tolerance.

1 Dataset

The dataset consists of over 25,247 anonymized credit card customer records with monthly behavioral variables spanning credit limits, bill amounts, payment amounts, and repayment history. The goal is to design a classification system that flags high-risk customers based on historical payment behavior. Predicting the variable `next_month_default` (1 = Default, 0 = No Default), the model enables the bank to mitigate losses by adjusting credit exposure and deploying early warnings.

Features Overview

The dataset includes both numerical and categorical features:

- Record of over 25,247 customers.
- Demographic Features: `sex`, `education`, `marriage`, and `age`.
- Credit Profile: `LIMIT_BAL` (credit limit), `pay_0` to `pay_6` (monthly payment delay history), `Bill_amt1` to `Bill_amt6` (monthly billed amount), `pay_amt1` to `pay_amt6` (monthly paid amount).

Target

- `next_month_default`: A binary label where `1` indicates the customer defaulted in the following month, and `0` indicates no default.

Class Imbalance

The dataset is highly imbalanced, with the majority of customers labeled as non-defaulters. To address this, Synthetic Minority Over-sampling Technique (SMOTE) was used to balance the class distribution before model training.

Validation Dataset

The validation dataset has the same feature structure as the training data but does not contain the target variable. It was used to generate final production-grade predictions after model selection and threshold tuning.

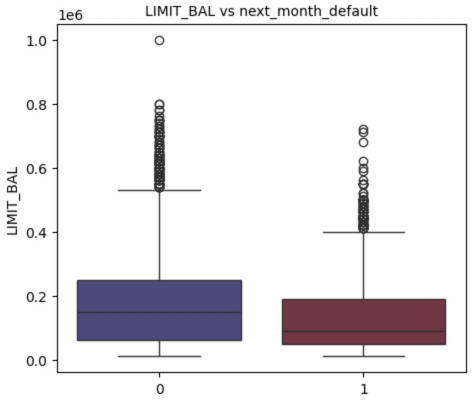
2 Exploratory Data Analysis (EDA)

EDA was conducted to understand variable distributions, detect trends, uncover data quality issues, and identify financially meaningful patterns that correlate with default risk.
Observations:

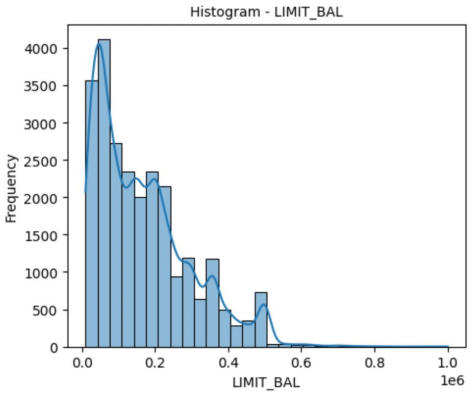
2.1 Univariate Analysis

Univariate plots such as histograms, boxplots, and violin plots were used to inspect distributions:

- `LIMIT_BAL` showed a right-skewed distribution with a high concentration of customers around mid-range limits, and some outliers at the upper end.

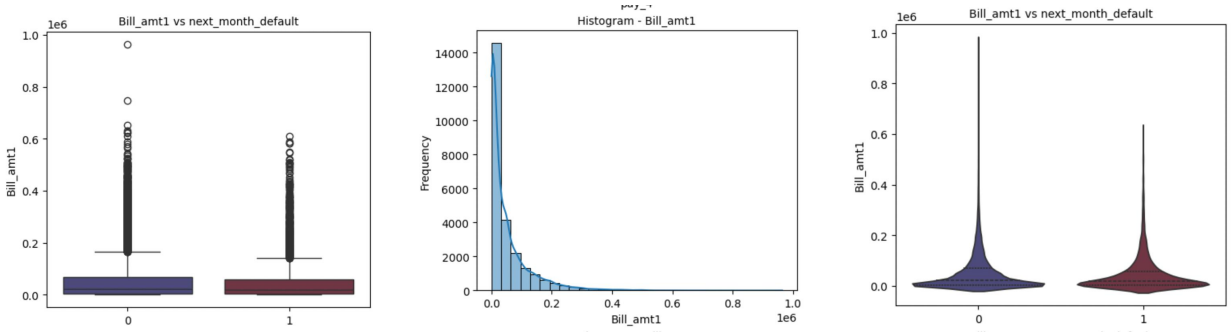


(a) Boxplot of `LIMIT_BAL`



(b) Histogram of `LIMIT_BAL`

- `age` was normally distributed but with a few younger customers showing more variability in repayment.
- Monthly bill (`Bill_amt`) and payment (`pay_amt`) amounts also exhibited heavy right skew.



(a) Boxplot of `Bill_amt1` (b) Histogram of `Bill_amt1` (c) Violin plot of `Bill_amt1`

Figure 2: Univariate distribution plots for key variables

2.2 Bivariate Analysis

Bivariate analysis was performed to examine the relationship between pairs of variables, primarily between independent features and the target variable `next_month_default`. This helps in identifying which variables are most influential in predicting defaults, and how different segments behave.

- Categorical vs Target: Default rates varied significantly across different values of categorical variables such as `sex`, `education`, and `marriage`. Defaulters were more frequent among customers with lower education levels and unmarried status.
  - Male customers exhibited a marginally higher default rate compared to females.
  - Default probability was inversely related to education level: customers with graduate degrees had lower default rates, while those with only high school education or unspecified categories had noticeably higher default risk.
  - Married individuals generally showed slightly lower default rates, potentially indicating more stable financial behavior.

- Monthly Payment Behavior: Scatter plots of `pay_amtX` vs `Bill_amtX` revealed that defaulters often paid much less than what was billed. This visual trend reinforces the importance of repayment ratio as a predictive feature.

Figure 3: Scatter plot of payment amount vs billed amount.

  - Non-defaulters tended to pay amounts that closely matched or exceeded their billed amounts, indicating financial responsibility.
  - Defaulters, however, often made consistently low payments regardless of bill volume — suggesting either poor cash flow or intentional underpayment.

- Payment Delay Impact: Grouped barplots using the `pay_0` feature showed that higher delay values are associated with increased default probability. Customers who had payment delays of 2 months or more were significantly more likely to default.

2.3 Correlation Analysis

From the correlation heatmap we can observe:
There is a high correlation between delayed payments (`pay_x`) and defaults, `bill_amt_x`, and `pay_amt_x` also have high correlation.

Key observations:

- Strong positive correlations were found among consecutive billing amounts (`Bill_amt1` to `Bill_amt6`) and among payment amounts (`pay_amt1` to `pay_amt6`). This indicates temporary stability in customer billing/payment behavior but also suggests potential multicollinearity.
- Payment delay features (`pay_0` to `pay_6`) were moderately correlated with each other, and `pay_1` showed the strongest linear correlation with the target variable `next_month_default` (0.31).
- Features like `LIMIT_BAL`, `age`, and demographic attributes had relatively low correlation with default, supporting the idea that behavioral signals are more predictive than demographic ones.
- Based on correlation thresholding (e.g., removing one of any pair with $|\rho| > 0.9$), redundant features were dropped to reduce noise and improve model generalization.

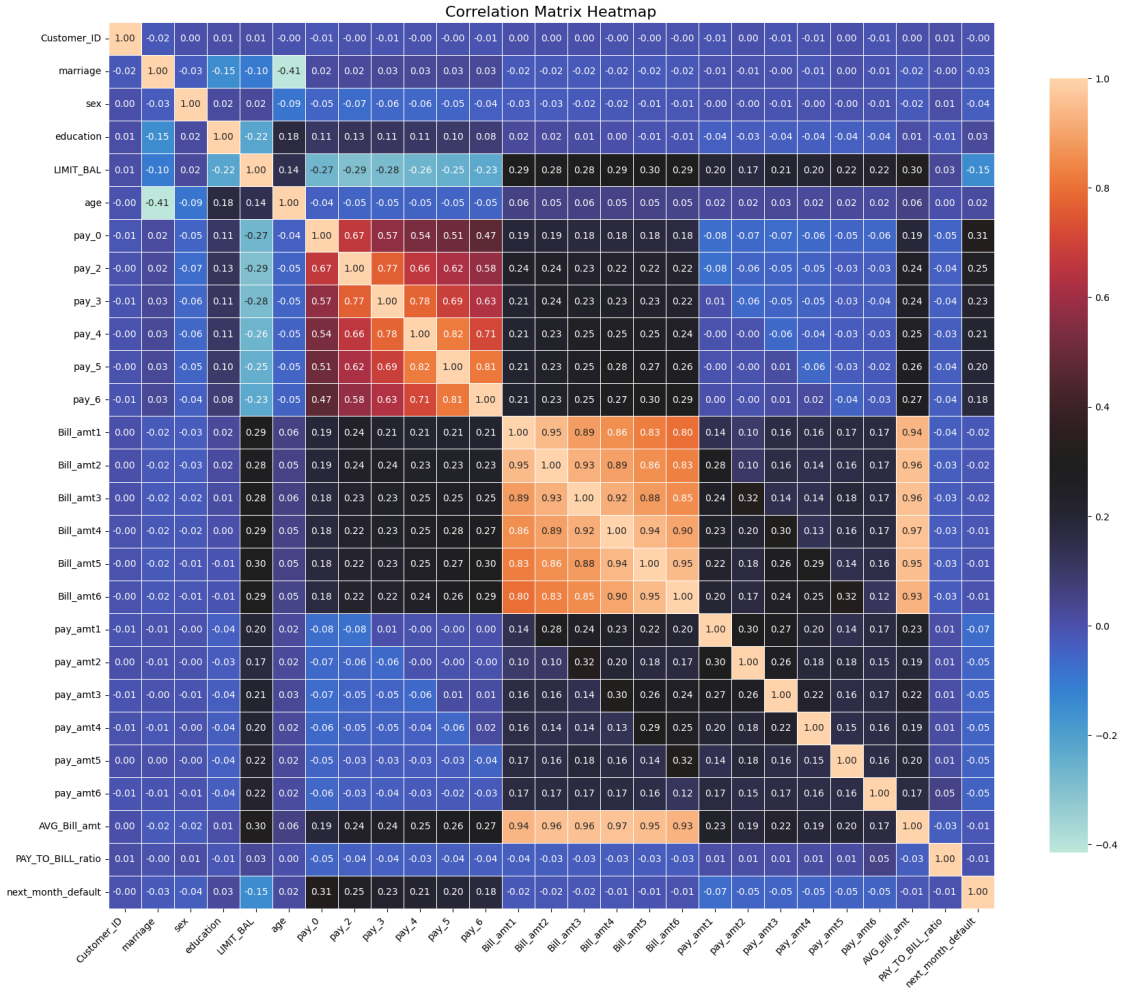


Figure 4: Correlation Matrix.

2.4 Bill and Payment Amount Patterns

- Bill Amounts: Although the absolute bill amount was not a strong predictor, defaulters tended to have more volatile monthly bills.
- Payment Amounts: Defaulters showed lower consistency in payment behavior, often underpaying compared to the billed amount.
- Repayment Ratios: Many defaulters had low `PAY_TO_BILL_ratio` values, indicating insufficient or delayed repayment.

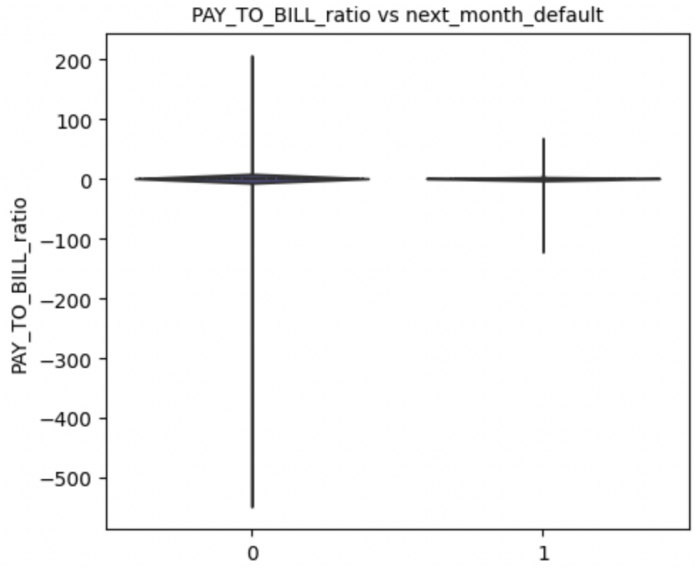


Figure 5: Violin plot of `PAY_TO_BILL_ratio` by default status. Defaulters tend to have lower repayment ratios.

2.5 Demographic Variables

- Sex: Male customers default rate is visibly higher than female.
- Education: Customers with lower education levels (especially category 3 = High School or below) were more likely to default than customers with graduation or uni education level, indicating a possible link between financial literacy and credit behavior.
- Marital Status: No strong trend emerged across marital categories, but single customers had marginally higher default risk.
- Age: Younger customers (aged 21–30) exhibited the highest default rates. Older customers (above 45) showed more conservative credit behavior.

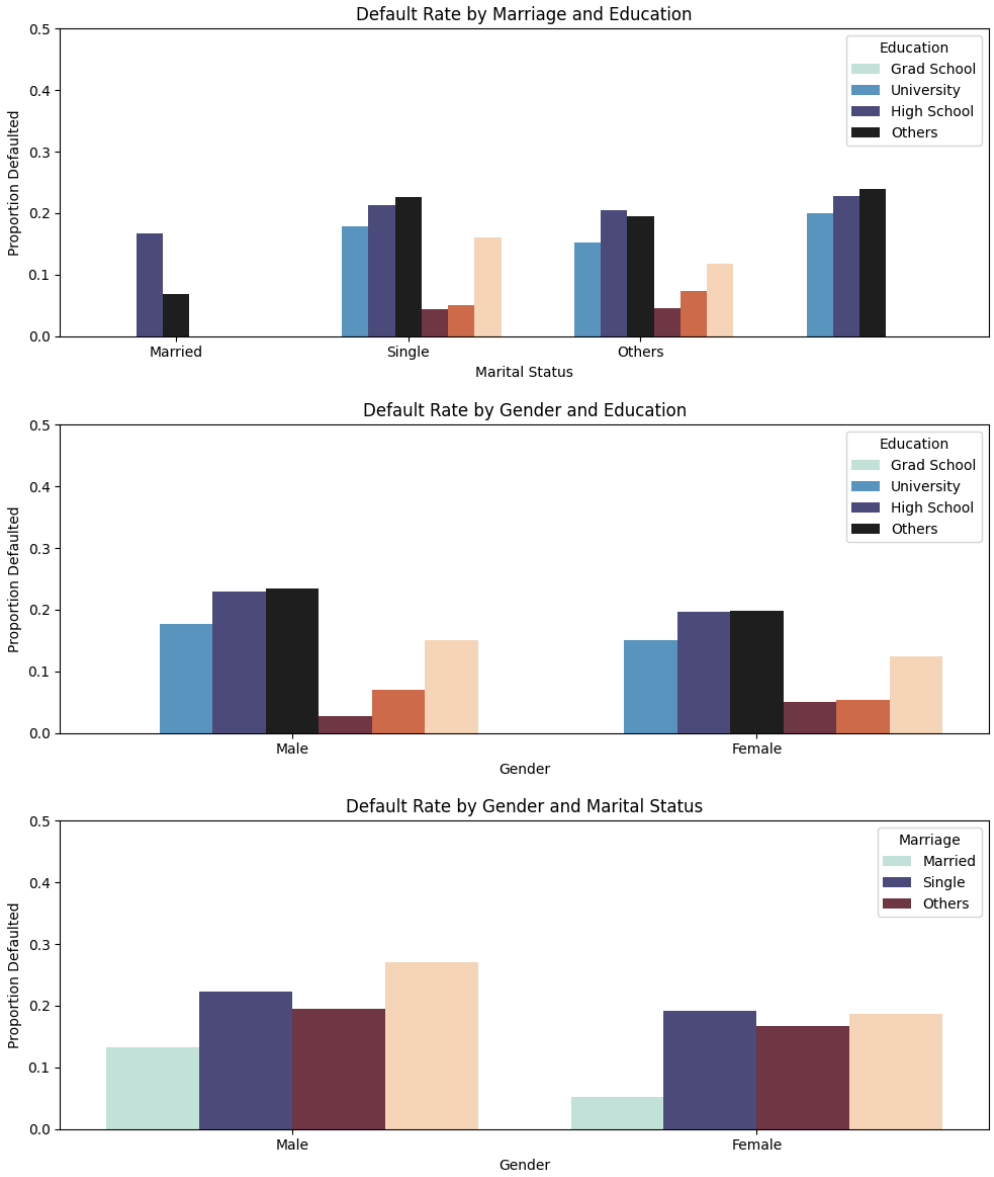


Figure 6: Default rate comparison across gender, age, education, and marital status.

2.6 Repayment Status (pay_0 to pay_6)

The repayment status variables had the strongest relationship with default:

- Defaulters consistently had higher values in `pay_x` variables (indicating delays).
- A trend of increasing delay over consecutive months was observed among defaulters.
- Non-defaulters mostly had `pay_x` values of -1 (fully paid on time) or 0 (partial but non-delinquent).

3 Data Preprocessing

A comprehensive data preprocessing pipeline was applied prior to modeling. This included handling categorical variables, transforming numerical features, and preparing the dataset for balanced training.

3.1 Handling Missing Values

The dataset had 126 missing values in `"age"` column. From EDA we can see that the distribution curve is highly skewed. So, using "median" strategy of `SimpleImputer` was best for filling in missing values of the age column.

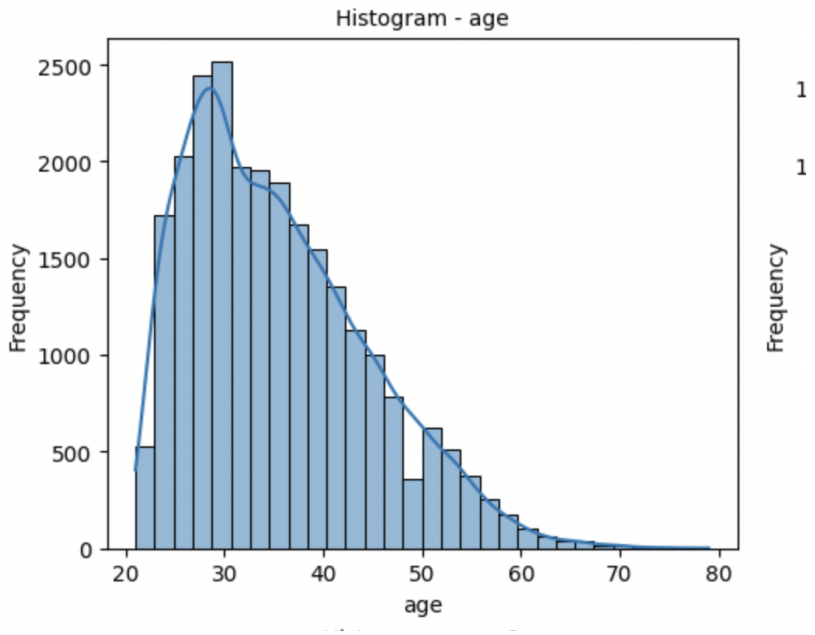


Figure 7: Distribution of "Age"

3.2 Normalization of Categorical Features

We observed that some categorical variables such as `education` and `marriage` contained undefined values.

- Education: Valid categories included `1 = Graduate School`, `2 = University`, `3 = High School`, and `4 = Others`. Values beyond this range (such as 5, 6 and 0) were recoded as `4 (Others)` to preserve the data.
- Marriage: Values like `1 = Married`, `2 = Single`, and `3 = Others` were valid. Unexpected values like `0` were reassigned to `3 (Others)`.

```
print(train_df.sex.value_counts())
print('-' * 50)
print(train_df.education.value_counts())
print('-' * 50)
print(train_df.marriage.value_counts())
print('-' * 50)
```

```
sex
1    15252
0     9995
Name: count, dtype: int64
--------------------------------------------------
education
2    11717
1     8988
3     4115
5      253
4      116
6       44
0       14
Name: count, dtype: int64
--------------------------------------------------
marriage
2    13441
1    11480
3      273
0       53
Name: count, dtype: int64
--------------------------------------------------
```

Figure 8: Value Count of Categorical Variables

3.3 Correlation Filtering

Highly correlated features: `Bill_amt1`, `Bill_amt2`, `Bill_amt3`, `Bill_amt4`, `Bill_amt5`, `pay_amt2`, `pay_amt3`, `pay_amt4`, `pay_amt5`, `pay_amt6` were dropped to reduce multicollinearity and avoid overfitting without significant information loss.

```
[ ] # dropping highly related columns
    drop_cols = [
        'Bill_amt1', 'Bill_amt2', 'Bill_amt3', 'Bill_amt4', 'Bill_amt5',
        'pay_amt2', 'pay_amt3', 'pay_amt4', 'pay_amt5', 'pay_amt6'
    ]

    train_df.drop(columns=drop_cols, inplace=True)
```

Figure 9: Code snippet

3.4 Encoding

We encoded categorical variables by using `OneHot Encoding`. This was necessarry to convert categorical variables into a set of binary (0/1) columns. Because the ML algorithms we are going to use cannot handle strings or categories directly. They expect numerical input.

3.5 Scaling, Splitting, and Balancing with SMOTE

To prepare the data for model training, we applied:

- **Train-Test Split:** The dataset was split into training and test sets in an 80:20 ratio using stratified sampling to preserve the proportion of default vs. non-default classes across both sets.
- **Standardization:** All numerical features were standardized using `StandardScaler`, transforming to have `0` mean and `unit` variance.
- **SMOTE** (Synthetic Minority Oversampling Technique): As the target variable `next_month_default` was highly imbalanced, with fewer defaulters than non-defaulters, we applied SMOTE to the training set. SMOTE helps improve recall and ensures the classifier is not biased toward the majority class.
- It is necessarry to note here that, train-test-split must pr performed before SMOTE. Because if we apply SMOTE before splitting, information from your test set leaks into your training set. This is called `data leakage` and can lead to overly optimistic model performance — our model learns patterns that it shouldn't know yet.

```python
from sklearn.preprocessing import StandardScaler
X = train_df.drop(columns=["next_month_default"])
y = train_df["next_month_default"]

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

smote = SMOTE()
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

Figure 10: SMOTE code snippet

Visualizing Class Balance:

The bar plots in Figure 11 show the class distribution before and after applying SMOTE. The left plot confirms the class imbalance in the original training data, while the right plot shows balanced classes after resampling.

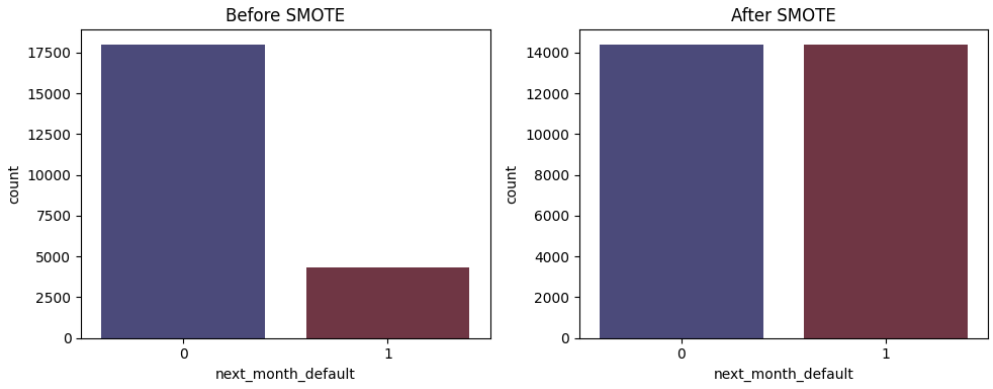


Figure 11: Class Distribution Before and After SMOTE

Quantitative Comparison:

A summary of the number of samples in each class before and after SMOTE is shown below:

Class	Before SMOTE	After SMOTE
No Default (0)	16352	16352
Default (1)	3845	16352

Table 1: Class Counts Before and After SMOTE Resampling

4 Financial Feature Engineering

To enhance model performance and improve interpretability, we engineered a set of financial features that capture customer behavior patterns more effectively than the raw variables alone. These derived features were created based on domain knowledge of credit risk, with a focus on repayment consistency, utilization, volatility, and delinquency.

- avg_utilization_ratio: Measures the average proportion of the credit limit used by the customer over six months. High values indicate aggressive credit use, which correlates with higher default risk.
- underpay_ratio: Captures the tendency of a customer to consistently pay less than their billed amount. It is computed as the proportion of months where payment was significantly lower than the bill.
- zero_payment_months: Counts the number of months where no payment was made despite having an outstanding bill. This helps flag chronic non-payers.
- repayment_consistency: Calculates the standard deviation of the `pay_x` values over six months. Higher values suggest erratic repayment behavior.
- bill_volatility and **payment_std**: Measure fluctuations in bill amounts and payment amounts respectively. High volatility often implies poor financial planning or unstable cash flow.
- max_delay and **avg_delay_value**: Represent the longest and average payment delay (in months) experienced by the customer. These are direct indicators of repayment discipline.
- delinquency_streak: Flags customers who had consecutive months of overdue payments, reflecting prolonged financial stress.
- avg_payment_ratio: Represents the ratio of actual payment to billed amount, averaged over six months. Low values are indicative of habitual underpayment.
- total_underpaid_amount and **total_overpaid_amount**: Capture cumulative payment deviations from expected amounts, helping the model understand payment surplus or deficit patterns.
- monthly_payment_range, **avg_payment_amount**, and **avg_bill_amount**: Summarize the range and average of billing/payment behavior to enhance model stability across varying customer profiles.

5 Modeling Strategy

We trained many models like `Logistic Regression`, `Decision Tree`, `Random Forest`, `XGBoost`, `LightGBM`. This section provides a detailed walkthrough of each modeling phase including model training, hyperparameter tuning, threshold optimization, ensemble learning, and final evaluation.

5.1 Model Training

These are the metrics obtained fro each model trained:

```
Logistic Regression
F2 Score: 0.5596508244422891
              precision    recall  f1-score   support

           0       0.90      0.82      0.86      4088
           1       0.44      0.60      0.51       962

    accuracy                           0.78      5050
   macro avg       0.67      0.71      0.68      5050
weighted avg       0.81      0.78      0.79      5050
```

```
Decision Tree Performance on Test Set:
Accuracy: 0.7928712871287129
F1 Score: 0.4872549019607843
F2 Score: 0.5044660982541616
ROC AUC: 0.760766387906799
```

```
Random Forest
F2 Score: 0.45601204560120456
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      4088
           1       0.53      0.44      0.48       962

    accuracy                           0.82      5050
   macro avg       0.70      0.67      0.69      5050
weighted avg       0.81      0.82      0.81      5050
```

```
XGBoost
F2 Score: 0.4172229639519359
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      4088
           1       0.58      0.39      0.47       962

    accuracy                           0.83      5050
   macro avg       0.72      0.66      0.68      5050
weighted avg       0.81      0.83      0.82      5050
```

```
LightGBM
F2 Score: 0.48226643598615915
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      4088
           1       0.57      0.46      0.51       962

    accuracy                           0.83      5050
   macro avg       0.73      0.69      0.71      5050
weighted avg       0.82      0.83      0.83      5050
```

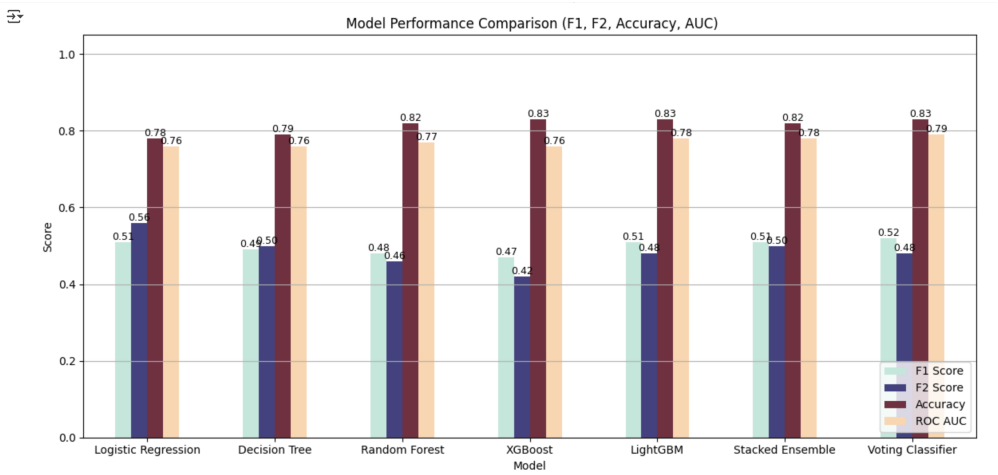


Figure 14: Comparision Bar chart of baseline models trained

5.2 Threshold Optimization

Rather than relying on the default 0.5 threshold, we swept thresholds from 0.1 to 0.9 and evaluated each using F1, F2, and Accuracy. Table 2 shows the thresholds that gave the best F2-score for each model.

Table 2: Thresholds Optimized for F2-Score

Model	Threshold	F1 Score	F2 Score	Accuracy	ROC AUC
Logistic Regression	0.37	0.417	0.508	0.579	0.761
Random Forest	0.38	0.47	0.60	0.69	0.78
XGBoost	0.30	0.48	0.53	0.77	0.76
LightGBM	0.30	0.50	0.57	0.78	0.78

We did not used Threshold Optimization on Decision Tree because our F2 Score is stuck at 0.412 across all thresholds, It suggests that model isn't capturing minority class singnals well, despite SMOTE and `class_weight='balanced'`.
So we use Grid Search CV for hyperparameter tuning of decision tree.

```
[ ] # best_dt = DecisionTreeClassifier(class_weight='balanced', max_depth=None, random_state=42)
    # best_dt.fit(X_train_smote, y_train_smote)

    # y_prob_dt = best_dt.predict_proba(X_test)[:, 1]
    # threshold_metrics = evaluate_thresholds(y_test, y_prob_dt)
    # print(threshold_metrics)
```

```
   Threshold  Accuracy  F1 Score  F2 Score  ROC AUC
0        0.3     0.725     0.378     0.412    0.615
1        0.4     0.725     0.378     0.412    0.615
2        0.5     0.725     0.378     0.412    0.615
3        0.6     0.725     0.378     0.412    0.615
4        0.7     0.725     0.378     0.412    0.615
5        0.8     0.725     0.378     0.412    0.615
6        0.9     0.725     0.378     0.412    0.615
```

Figure 15: Thresholds of Decision Tree

```
param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

grid_search_dt = GridSearchCV(
    estimator=DecisionTreeClassifier(class_weight='balanced', random_state=42),
    param_grid=param_grid,
    scoring=f2_scorer,
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search_dt.fit(X_train_smote, y_train_smote)
best_dt = grid_search_dt.best_estimator_
y_pred = best_dt.predict(X_test)
y_prob = best_dt.predict_proba(X_test)[:, 1]
```

Figure 16: Code Snippet for GridSearchCV

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Accuracy: 0.7279207920792079
F1 Score: 0.3737465815861404
F2 Score: 0.4035433070866142
ROC AUC: 0.6125595526280457
```

Figure 17: Decision Tree evaluation metrics after grid search cv

5.3 Ensemble Learning

To further improve predictive performance, we constructed ensemble models:

- **Stacking Classifier:** Combined predictions from the top four base learners (LogReg, DT, LightGBM), with a logistic regression meta-model. This model achieved the best F2-score i.e. 0.60 on the test set after threshold tuning but low accuracy of 0.57.
- **Voting Classifier:** A soft-voting ensemble averaged the predicted probabilities. It performed comparably but slightly worse than stacking in F2-score.

5.5 Final Evaluation

Table 3 summarizes the test-set performance.

Table 3: Final Model Evaluation Metrics on Test Set

Model	F1 Score	F2 Score	Accuracy	ROC AUC
Stacked Ensemble	0.42	0.60	0.58	0.78
Voting Classifier	0.52	0.48	0.83	0.79
Tuned Logistic Regression	0.42	0.58	0.58	0.76
Tuned Decision Tree	0.37	0.40	0.73	0.61
Tuned Random Forest	0.47	0.60	0.69	0.78
Tuned XGBoost	0.48	0.53	0.77	0.76
Tuned LightGBM	0.51	0.57	0.78	0.78

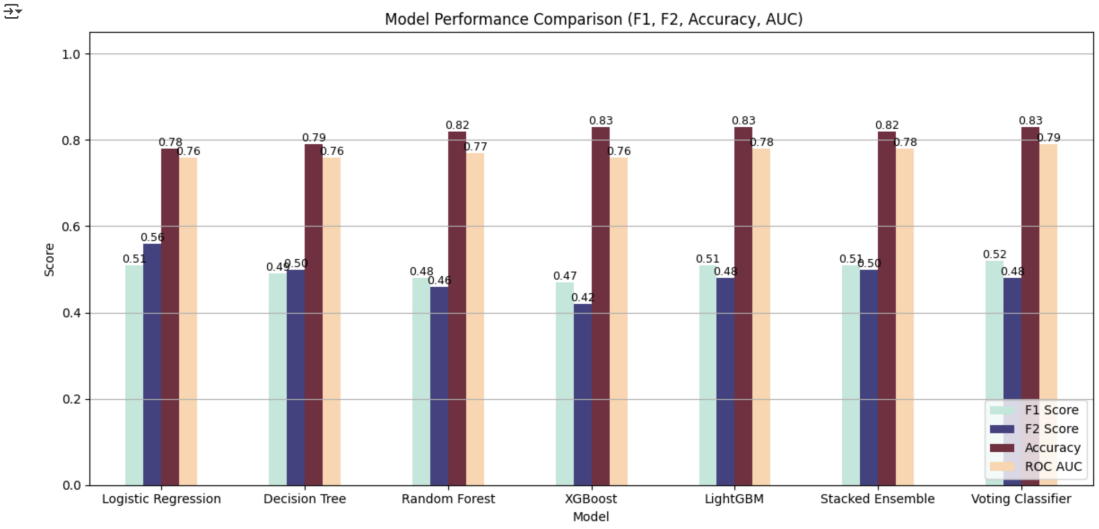


Figure 18: Comparision Chart Tuned Models

In the context of credit risk prediction, minimizing false negatives is more critical than minimizing false positives. A false negative corresponds to a risky customer who is predicted as non-defaulter, potentially resulting in financial loss to the credit institution.

To reflect this business priority, we chose the `F2-score` as our primary evaluation metric. F2-score places more emphasis on recall. This aligns with our goal of maximizing the detection of potential defaulters.

Therefore, all model selection, hyperparameter tuning, and threshold optimization steps were guided by the goal of maximizing F2-score while maintaining reasonable values for accuracy and F1-score.

The final recommended model we got was Tuned LightGBM, having nice F2 score along with Accuracy.

```python
# Recommend best model based on selected metrics
best_model_name, ranked_scores = recommend_best_model(tuned_results_rounded.reset_index())

print(" Recommended Best Model:", best_model_name)
best_f2 = tuned_results_rounded.loc[best_model_name, "F2 Score"]
best_acc = tuned_results_rounded.loc[best_model_name, "Accuracy"]
print(f" Best Model F2 Score: {best_f2}")
print(f" Best Model Accuracy: {best_acc}")
```

```
Recommended Best Model: Tuned LightGBM
Best Model F2 Score: 0.57
Best Model Accuracy: 0.78
```

Figure 19: Best Model Code snippet

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

           0       0.90      0.81      0.85      4088
           1       0.44      0.61      0.51       962

    accuracy                           0.78      5050
   macro avg       0.67      0.71      0.68      5050
weighted avg       0.81      0.78      0.79      5050
```

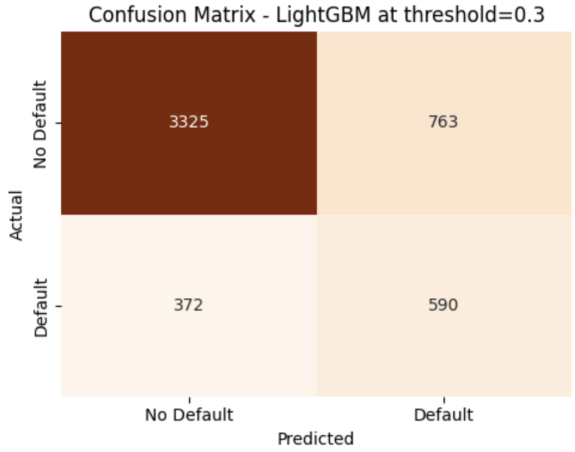


Figure 20: Best Model Report

6 Validation Predictions

On the unlabeled validation dataset, predictions were generated using the final tuned model and stored in a CSV format with two columns: `Customer`, `next_month_default`. The Best Model was applied to the unlabeled validation dataset after applying the same preprocessing and threshold.

```
threshold = 0.30
y_val_prob = best_lgbm.predict_proba(X_validate_scaled)[:, 1]
y_val_pred = (y_val_prob >= threshold).astype(int)

val["next_month_default"] = y_val_pred

val[["Customer_ID", "next_month_default"]].to_csv("submission_23112073.csv", index=False)
print("Predictions saved to submission_23112073.csv")
```

```
Predictions saved to submission_23112073.csv
```

Figure 21: Validation Code

7 Selection of F2-Score Over Other Metrics

Several evaluation metrics were considered during model development, including Accuracy, Precision, Recall, F1-score, F2-score, and ROC-AUC. However, after reviewing the business context and implications of misclassification, the F2-score was prioritized as the primary model selection criterion.

- `Accuracy`: due to class imbalance a model could achieve high accuracy by simply predicting the majority class (non-defaulters), while missing most actual defaulters.
- `Precision` measures how many predicted defaulters were truly defaulters. But precision alone does not penalize the model for missing actual defaulters, which is costlier in credit risk applications.
- `Recall` emphasizes capturing all actual defaulters, which is more aligned with our goal. However, recall alone ignores how many false positives the model generates.
- `F1-score` balances precision and recall equally, but in our domain, recall is more valuable. Missing a defaulter (false negative) is riskier than flagging a safe customer (false positive).
- `F2-score`, which weights recall more heavily than precision, was therefore selected. It ensures that the model is more sensitive to defaulters, reducing the chance of financial exposure due to missed risks.
- `ROC-AUC` was used as a supporting metric to evaluate the model's overall prediction ability, but F2-score guided actual classification threshold tuning and model choice.

Ultimately, F2-score was the most suitable metric. All model tuning and threshold optimization were conducted with the objective of maximizing F2-score while maintaining acceptable levels of F1-score and accuracy.

8 Conclusion

This model predicted 27.01% as defaulters which is greater than the actual defaulters (19.04%) train dataset had (real world). This model provides a risk-sensitive, interpretable scoring system that supports Bank A's early-warning strategy. By combining robust EDA, financial engineering, and tuned classification pipelines, the solution aligns technical accuracy with business impact.