Finance Club

Open Project Summer 2025

Project 2 - Credit Card Default Prediction

-Prakhar Sharma

23113114

1. Introduction

The aim of this study is to exploit supervised machine learning algorithms to identify the key drivers that determine the likelihood of credit card default. Credit card defaults pose a major risk to banks and financial institutions, making it important to identify customers who are likely to default in advance. In this project, we are provided with anonymized historical data of over 30,000 credit card users, along with a target variable — default.payment.next.month — which shows whether a customer defaulted on their payment in the next billing cycle.

The main goal is to build a machine learning model that can accurately predict which customers are at risk of defaulting. But beyond just making predictions, the model should also help us understand the key factors that contribute to defaults. This would allow banks to take timely actions like adjusting credit limits, flagging high-risk customers early, and managing credit exposure more effectively.

2. Exploratory Data Analysis (EDA)

2.1 Feature Analysis

The dataset provided consists of 30,000 observations that represent distinct credit card clients. Each observation has 26 attributes that contain information on default payments, history of payments, etc.

The first group of variables contains information about client personal information:

1 ID – ID of each client

2 LIMIT_BAL - Credit limit assigned to the customer (in currency units)

3 education – Level of education

4 marriage – marital status

5 sex - Gender

6 age – Age in years

The next group contains information about delay of past payment, amount of bill statement and amount of previous payments with attributes like:

1 Pay_m (Pay_0 to 6) – Pay_m represents the payment status in the most recent month m.

2 Bill_amt_m (Bill_amt1 to 6) - Total bill amount at the end of month m

3 Pay_amt_m (Pay_amt1 to 6) - Payment amount made in month m towards the bill generated in month m-1.

The last variable is the one to be predicted:

1 next_month_default - Target variable: 1 if customer defaulted next month , 0 otherwise

In order to look at how data is presented, some basic code like shape, describe were used.

```
# Preview
print("Train shape:", train_df.shape)
print("Validation shape:", val_df.shape)
print(train_df.head())
print(val_df.head())
train_df.describe()
```

```
Train shape: (25247, 27)
Validation shape: (5016, 26)
   Customer_ID  marriage  sex  education  LIMIT_BAL   age  pay_0  pay_2  \
0         5017         2    0          2      60000  25.0      2      2
1         5018         2    1          1     290000  24.0      0      0
2         5019         1    0          2     180000  63.0      0      0
3         5020         1    1          2     210000  43.0      0      0
4         5021         2    0          1     280000  32.0     -2     -2

   pay_3  pay_4  ...  Bill_amt6  pay_amt1  pay_amt2  pay_amt3  pay_amt4  \
0      2      0  ...   20750.63   2000.21      0.00   1134.85   1821.78
1     -2     -2  ...    1350.30      0.00      0.17      0.00   2700.10
2      0      0  ...   52991.51   2086.94   2199.99   1845.66   2000.35
3      0      0  ...   76945.47   3348.07   3380.91   3400.45   2683.97
4     -2     -2  ...       1.35    999.78   3186.27  45027.78   2100.09

   pay_amt5  pay_amt6  AVG_Bill_amt  PAY_TO_BILL_ratio  next_month_default
0   1500.03   1500.24      41511.50               0.03                   0
1      0.00   1349.72       2534.50               0.27                   0
2   1923.00   1999.78      50422.00               0.04                   0
3   2744.00   2892.10      86229.50               0.04                   0
4      0.01      0.27      11814.33               0.72                   0

[5 rows x 27 columns]
   Customer_ID  marriage  sex  education  LIMIT_BAL  age  pay_0  pay_2  pay_3  \
0            1         1    1          2     220000   32      0      0      0
1            2         2    0          1     350000   35     -1     -1     -1
2            3         2    1          1     310000   39      0      0      0
3            4         1    0          2      20000   47      0      0      0
4            5         2    1          2     500000   30      0      0      0

   pay_4  ...  Bill_amt5  Bill_amt6  pay_amt1  pay_amt2  pay_amt3  pay_amt4  \
0      0  ...   17831.13   15670.47   2000.03   3999.90   1419.80   1999.97
1      0  ...   10832.78    2261.45  33891.01  16267.19   4026.80    234.10
2      0  ...  240520.57  246524.45  11026.94  10499.83  14000.32  10000.12
3      2  ...   15040.17   14749.97   1200.00   2799.83      0.14   1499.93
4      0  ...   69054.15   64841.30  25463.94  43095.31   7521.96   9065.17

   pay_amt5  pay_amt6  AVG_Bill_amt  PAY_TO_BILL_ratio
0   3000.21  30788.71      23456.33               0.31
```

Now the following observations were made:

1. The column pay_0 was renamed to pay_1 to avoid confusion.
2. In the columns education and marriage there are some undocumented categories.
3. The column Pay have minimum in -2 and max 8.
4. Age attribute has missing values.

2.2 Data Cleaning

The presence of errors in dataset can be addressed in two ways:

1. Deleting of the rows associated with errors.
2. With correction of the wrong attributes.

In our case first method is applied.

```
[15]: # Keep only valid values in 'marriage' (1,2,3)
      train_df = train_df[train_df['marriage'].isin([1, 2, 3])]

      # Keep only valid values in 'education' (1,2,3,4)
      train_df = train_df[train_df['education'].isin([1, 2, 3, 4])]
```

Next imputing the missing age values with the median.

```
# Impute missing AGE values with median
median_age = train_df["age"].median()
train_df["age"].fillna(median_age, inplace=True)
```

Next renaming the column pay_0 with pay_1 and dropping column Customer_ID as it has no particular significance and no direct relation with customer default.

```
[12]: train_df = train_df.drop(columns=["Customer_ID"])
      train_df = train_df.rename(columns={"pay_0": "pay_1"})
      val_df = val_df.drop(columns=["Customer_ID"], errors='ignore')
      val_df = val_df.rename(columns={"pay_0": "pay_1"})
```

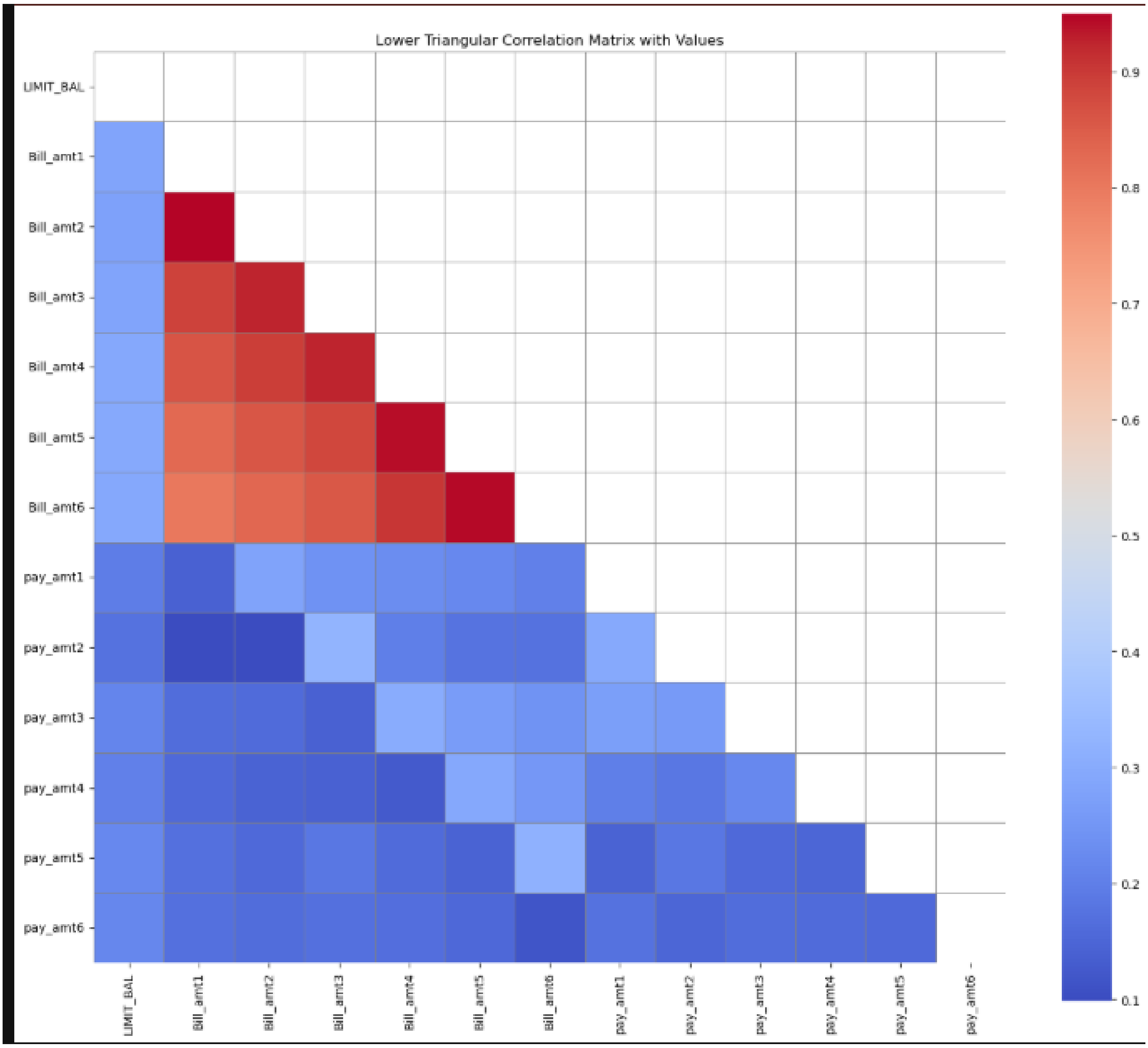
Next there were some duplicate values that were removed.

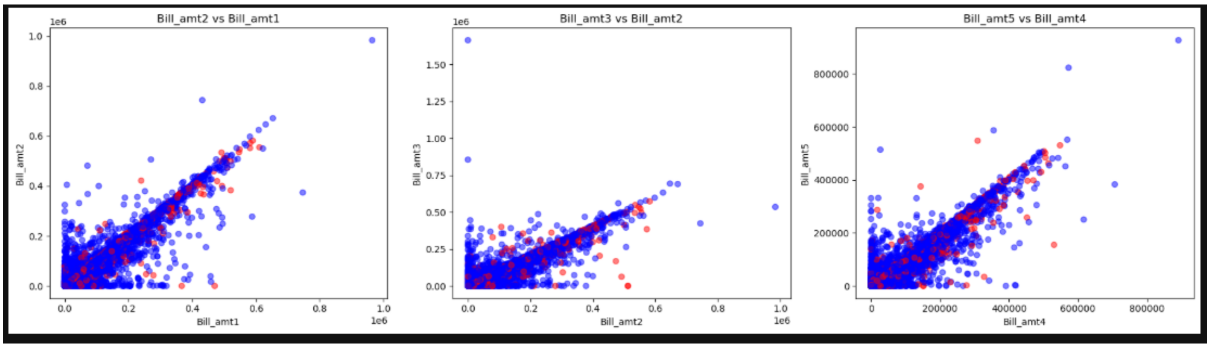
```
[299]: X_train.duplicated().sum()

[299]: 792

•[300]: X_train = X_train.drop_duplicates()
        y_train = y_train.loc[X_train.index]
```

2.3 Correlation Matrix

Another relevant point which could affect the classification performances is the correlation among features: the presence of strongly correlated features may lead to a decline in the performances of some classification algorithms which assume that the predictors are all independent. But since we will build an XGboost model.

But anyway, here is the correlation matrix:





The above shown scatterplot show that there is a significance correlation between some features

3.Data Preprocessing

3.1 One-Hot Encoding for categorical variables

Categorical variables such as sex, marriage, education are turned into one-hot variables. It is a representation of categorical variables as binary vectors.

```
[69]: # One-hot encode categorical columns
categorical_cols = ['sex', 'marriage', 'education']

train_df = pd.get_dummies(train_df, columns=categorical_cols, drop_first=True)
val_df = pd.get_dummies(val_df, columns=categorical_cols, drop_first=True)
```

3.2 Separating target features

Separating the target variable from other attributes to work on them.

```
[288]: # Separate target and features
y_train = train_df['next_month_default']
X_train = train_df.drop(columns=['next_month_default'])

#y_val = val_df['next_month_default']
X_val = val_df.drop(columns=['next_month_default'])
```

3.3Feature Extraction

Extracting important features like
max_delay – Maximum payment delay
avg_delay – Average payment delay
delay_count – count of months with delay
avg_bill_amt – Average bill amount
credit_utilization – Utilization of the credit services and some others.
Which would prove to be advantageous when building the model.

3.4 Handling Class Imbalance

Since the dataset was imbalanced with significantly fewer defaulters compared to non-defaulters, I used the scale_pos_weight parameter in the XGBoost model to handle this issue. This parameter was set as the ratio of non-defaulters to defaulters in the training data, so the model would give more weight to predicting defaulters correctly. Without this adjustment, the model might just predict most customers as non-defaulters to achieve high accuracy, while actually performing poorly in identifying risky customers. By using class weighting, the model became more focused on catching potential defaulters, which is important in a credit risk scenario where missing them can lead to major financial losses.

```
[304]: # Use the same X_train and y_train
scale_pos_weight = y_train.value_counts()[0] / y_train.value_counts()[1]

from xgboost import XGBClassifier
model_weighted = XGBClassifier(scale_pos_weight=scale_pos_weight, random_state=42)
model_weighted.fit(X_train, y_train)
```

```
[304]: XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

2.1 Train-Test Split

Now the training data is further slitted into train and test data.

```
[305]: from sklearn.model_selection import train_test_split
X_train_train, X_train_test, y_train_train, y_train_test = train_test_split(
    X_train, y_train, test_size=0.2, stratify=y_train, random_state=42
)
```

4 Models

For this problem, I chose to focus on XGBoost and LightGBM because both are powerful gradient boosting algorithms that perform exceptionally well on structured, tabular datasets like the one provided. These models are capable of capturing complex non-linear relationships, handling missing values, and managing large feature sets without the need for extensive preprocessing such as feature scaling or dimensionality reduction. Given that the dataset had class imbalance and several engineered behavioral features, tree-based ensemble methods like XGBoost and LightGBM were well-suited to extract meaningful patterns. Additionally, they offer built-in support for class weighting, fast training, and high predictive performance, making them ideal candidates for a high-stakes classification task like credit risk prediction, where recall is critical and interpretability is also important.

4.1 XGBoost

I chose XGBoost as the main model for this project because it's highly effective for structured/tabular data like credit card transactions. XGBoost, which stands for Extreme Gradient Boosting, is an ensemble learning method based on decision trees. It works by building trees sequentially, where each new tree tries to correct the errors made by the previous ones. One major advantage of using XGBoost is that it doesn't require feature scaling or dimensionality reduction, as it can handle high-dimensional data and correlated features effectively. It's also robust to missing values and captures non-linear relationships and feature interactions very well. In this case, it performed particularly well in identifying defaulting customers, especially when combined with class imbalance handling techniques like scale_pos_weight.

4.2 LightGBM

I also experimented with LightGBM (Light Gradient Boosting Machine), which is another powerful gradient boosting framework designed for high performance and speed. Like XGBoost, it builds decision trees sequentially to minimize prediction errors, but it differs in the way it grows trees — LightGBM uses a leaf-wise growth strategy instead of level-wise, which often leads to better accuracy with less training time. Similar to XGBoost, LightGBM handles missing values, categorical features (with proper encoding), and high-dimensional data efficiently, without requiring feature scaling or dimensionality reduction. It's particularly known for being faster on large datasets. Although it performed reasonably well in this case, XGBoost slightly outperformed it in terms of F2-score, which was the primary metric of focus for this project.

5 Evaluation Metrices

In a classification scenario, the model can be evaluated by computing different metrics. In order to better understand these metrics could be useful to get some fundamentals:

• True Positive (TP): samples for which the prediction is positive and the true class is positive • False Positive (FP): samples for which the prediction is positive but the true class is negative • True Negative (TN): samples for which the prediction is negative and the true class is negative • False Negative (FN): samples for which the prediction is negative but the true class is positive.

Some of the most popular metrics are:

• Accuracy: ratio of correct predictions over the total number of data points classified \begin{equation} Accuracy = \frac{# correctly classified samples} {total number of samples tested}=\frac{TP+TN}{TP+FP+TN+FN} \end{equation}

• Precision: measures the fraction of correct classified instances among the ones classified as positive. Precision is an appropriate measure to use when the aim is to minimize false positives. \begin{equation} Precision(c) = \frac{# samples correctly assigned to class c} {# of samples assigned to class c}=\frac{TP}{TP+FP} \end{equation}

• Recall: it measures how many of the actual positives a model capture through labelling it as True Positive.It is an appropriate score when the aim is to minimize false negatives \begin{equation} Recall(c) = \frac{# samples correctly assigned to class c} {# of samples actually belonging to c}=\frac{TP}{TP+FN} \end{equation}

• F1-score: is the harmonic mean of the precision and recall.

• F2-score: is a metric that balances precision and recall, but gives more weight to recall.

6 Evaluation

Here are the results obtained for XGBoost:

```
[306]: from sklearn.metrics import precision_score, recall_score, f1_score, fbeta_score

y_pred = model_weighted.predict(X_train_test)

precision = precision_score(y_train_test, y_pred)
recall = recall_score(y_train_test, y_pred)
f1 = f1_score(y_train_test, y_pred)
f2 = fbeta_score(y_train_test, y_pred, beta=2)

print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")
print(f"F2 Score:  {f2:.4f}")
```

```
Precision: 0.7184
Recall:    0.9279
F1 Score:  0.8098
F2 Score:  0.8768
```

And here are the same results for LGBM:

```
[308]: from sklearn.metrics import precision_score, recall_score, f1_score, fbeta_score

y_pred_lgbm = lgbm_model.predict(X_train_test)

precision = precision_score(y_train_test, y_pred_lgbm)
recall = recall_score(y_train_test, y_pred_lgbm)
f1 = f1_score(y_train_test, y_pred_lgbm)
f2 = fbeta_score(y_train_test, y_pred_lgbm, beta=2)

print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")
print(f"F2 Score:  {f2:.4f}")

Precision: 0.5550
Recall:    0.7962
F1 Score:  0.6541
F2 Score:  0.7326
```

And here is an insightful table from which we can select our desired precision and f2 score value:

```
[373]: from sklearn.metrics import precision_score, recall_score, fbeta_score

y_probs = final_model.predict_proba(X_train)[:, 1]

for thresh in np.arange(0.4, 0.8, 0.01):
    y_thresh = (y_probs >= thresh).astype(int)
    precision = precision_score(y_train, y_thresh)
    recall = recall_score(y_train, y_thresh)
    f2 = fbeta_score(y_train, y_thresh, beta=2)
    print(f"Threshold: {thresh:.2f} | Precision: {precision:.3f} | Recall: {recall:.3f} | F2: {f2:.3f}")

Threshold: 0.40 | Precision: 0.598 | Recall: 0.973 | F2: 0.865
Threshold: 0.41 | Precision: 0.610 | Recall: 0.969 | F2: 0.867
Threshold: 0.42 | Precision: 0.621 | Recall: 0.967 | F2: 0.870
Threshold: 0.43 | Precision: 0.633 | Recall: 0.962 | F2: 0.872
Threshold: 0.44 | Precision: 0.644 | Recall: 0.957 | F2: 0.873
Threshold: 0.45 | Precision: 0.657 | Recall: 0.953 | F2: 0.874
Threshold: 0.46 | Precision: 0.670 | Recall: 0.948 | F2: 0.875
Threshold: 0.47 | Precision: 0.682 | Recall: 0.942 | F2: 0.875
Threshold: 0.48 | Precision: 0.693 | Recall: 0.936 | F2: 0.875
Threshold: 0.49 | Precision: 0.702 | Recall: 0.930 | F2: 0.873
Threshold: 0.50 | Precision: 0.712 | Recall: 0.923 | F2: 0.871
Threshold: 0.51 | Precision: 0.722 | Recall: 0.917 | F2: 0.870
Threshold: 0.52 | Precision: 0.732 | Recall: 0.909 | F2: 0.867
Threshold: 0.53 | Precision: 0.743 | Recall: 0.899 | F2: 0.863
Threshold: 0.54 | Precision: 0.754 | Recall: 0.893 | F2: 0.861
Threshold: 0.55 | Precision: 0.762 | Recall: 0.887 | F2: 0.859
Threshold: 0.56 | Precision: 0.774 | Recall: 0.879 | F2: 0.855
Threshold: 0.57 | Precision: 0.784 | Recall: 0.868 | F2: 0.850
Threshold: 0.58 | Precision: 0.794 | Recall: 0.860 | F2: 0.846
Threshold: 0.59 | Precision: 0.801 | Recall: 0.848 | F2: 0.838
Threshold: 0.60 | Precision: 0.809 | Recall: 0.837 | F2: 0.831
Threshold: 0.61 | Precision: 0.818 | Recall: 0.826 | F2: 0.825
Threshold: 0.62 | Precision: 0.827 | Recall: 0.814 | F2: 0.816
Threshold: 0.63 | Precision: 0.835 | Recall: 0.802 | F2: 0.809
Threshold: 0.64 | Precision: 0.845 | Recall: 0.793 | F2: 0.803
Threshold: 0.65 | Precision: 0.853 | Recall: 0.780 | F2: 0.793
Threshold: 0.66 | Precision: 0.862 | Recall: 0.767 | F2: 0.784
Threshold: 0.67 | Precision: 0.869 | Recall: 0.750 | F2: 0.771
Threshold: 0.68 | Precision: 0.875 | Recall: 0.734 | F2: 0.759
Threshold: 0.69 | Precision: 0.880 | Recall: 0.723 | F2: 0.749
Threshold: 0.70 | Precision: 0.888 | Recall: 0.710 | F2: 0.739
Threshold: 0.71 | Precision: 0.896 | Recall: 0.692 | F2: 0.725
Threshold: 0.72 | Precision: 0.900 | Recall: 0.677 | F2: 0.712
```

So according to the case that we are handling here(of credit card default prediction) I would recommend a high value of recall and f2 score about recall = 0.936 and F2 = 0.875. Since in credit card default prediction, recall is often considered more important than precision because missing a true defaulter can lead to significant financial losses for the bank. On the other hand, wrongly flagging a non-defaulter is less costly and can usually be managed through manual review or temporary credit restrictions. As a result, many credit risk models are designed to prioritize high recall, while still maintaining reasonable precision to avoid excessive false positives. To strike this balance, the F2-score is commonly used as an evaluation metric, since it places greater emphasis on recall, making it well-suited for risk-sensitive applications like this one.

7 Conclusion

In conclusion, this project aimed to develop a predictive model to identify credit card customers who are likely to default in the upcoming month. By performing detailed exploratory analysis, meaningful feature engineering, and testing advanced classification models like XGBoost and LightGBM, the final model was able to achieve a high F2-score, indicating strong recall with reasonable precision. This is especially important in the context of credit risk, where failing to identify a true defaulter can result in significant financial loss for the bank. The model's ability to flag high-risk customers in advance allows the bank to take proactive measures such as adjusting credit limits, triggering early warning systems, or initiating personalized customer outreach. These actions can help minimize default rates, reduce financial exposure, and improve overall portfolio health. The project not only highlights the value of machine learning in financial risk management but also demonstrates how data-driven decisions can directly contribute to better business outcomes.

- THE END -