FinCLub@IITR

# CREDIT-RISK MODELLING

Adesh Pal - 23323005

Credit Risk
['kre-dit 'risk]
The possibility of a loss resulting from a borrower's failure to repay a loan or meet contractual obligations.
Investopedia

## Introduction

Effective credit risk management is essential for the stability and profitability of financial institutions. In the rapidly evolving landscape of consumer credit, banks must move beyond reactive approaches and adopt forward-looking strategies to identify and mitigate risk proactively. Bank A, recognizing this need, seeks to enhance its credit risk framework by developing a Behaviour Score — a predictive model that flags customers likely to default on their credit card payments in the upcoming billing cycle.

This project leverages a rich dataset containing anonymized behavioral and financial information of over 30,000 credit card customers. The central objective is to build a classification model that can accurately predict the binary outcome variable: `default.payment.next.month`, indicating whether a customer defaults in the next month. By identifying high-risk customers in advance, the bank can take timely actions such as adjusting credit limits, initiating early warning alerts, or prioritizing collection efforts.

Beyond predictive accuracy, the model must also offer financial interpretability, enabling credit risk managers to understand key drivers of default. This transparency supports regulatory compliance, enhances trust in the model's decisions, and empowers the bank to develop data-driven, risk-based policies.

This report outlines the data exploration, preprocessing, model development, evaluation, and interpretability strategies employed to deliver a robust and actionable Behaviour Score system for Bank A.

## Project Flow

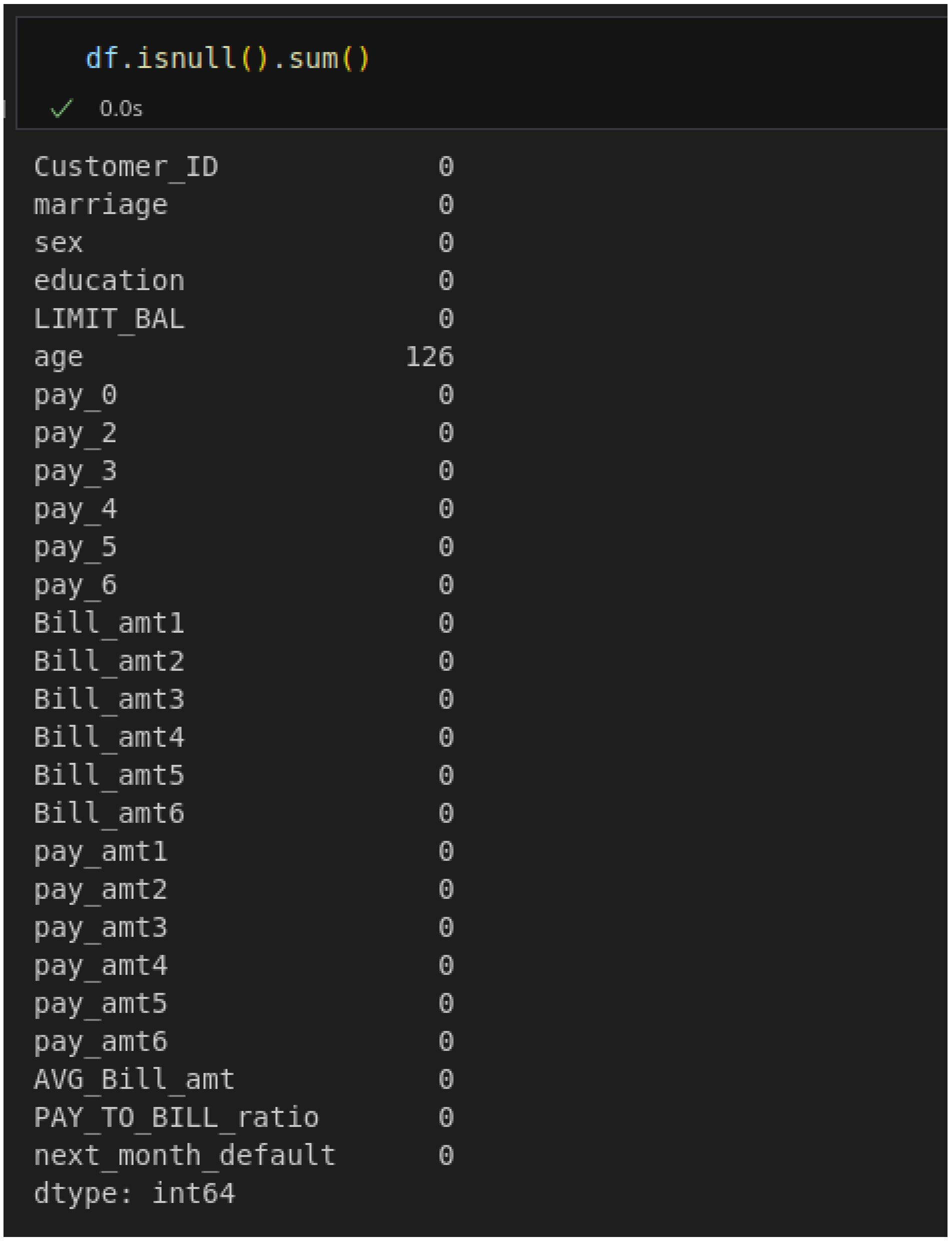
- Data Analysis & Exploration-
    1. Load and inspect dataset shape, column types, missing values.
    2. Analyzing Class distribution (default vs non-default), Feature distributions (e.g., payment history, credit limits), Correlation heatmap to identify multicollinearity etc.
    3. Visual exploration: Histograms, boxplots, and scatterplots for key variables.

- Data Preprocessing- Handle missing values. One hot encode categorical variables and address class imbalance(SMOTE).
- Feature Engineering- Create new features eg- Utilization ratio, repayment trend etc
- Feature Selection- Use correlation and variance analysis to drop irrelevant/redundant features.

- Model Training- Training models like Random Forest classifier, XGboost using k-fold cross validation.
- Model Evaluation- Use classification metrics like accuracy, precision, recall, f2 score, AUC-ROC curve and confusion matrix.

## Data Analysis and Exploration

1. Analysing Null values



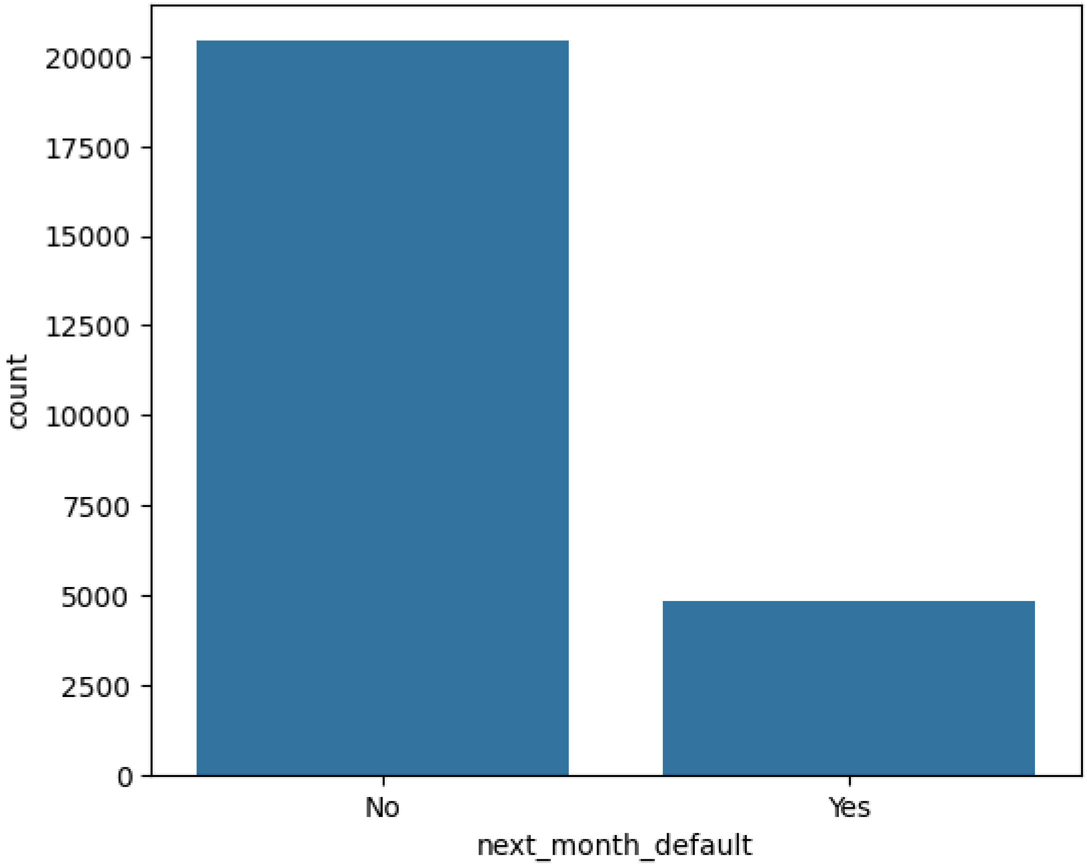
```
df.isnull().sum()
```

✓ 0.0s

```
Customer_ID            0
marriage               0
sex                    0
education              0
LIMIT_BAL              0
age                  126
pay_0                  0
pay_2                  0
pay_3                  0
pay_4                  0
pay_5                  0
pay_6                  0
Bill_amt1              0
Bill_amt2              0
Bill_amt3              0
Bill_amt4              0
Bill_amt5              0
Bill_amt6              0
pay_amt1               0
pay_amt2               0
pay_amt3               0
pay_amt4               0
pay_amt5               0
pay_amt6               0
AVG_Bill_amt           0
PAY_TO_BILL_ratio      0
next_month_default     0
dtype: int64
```

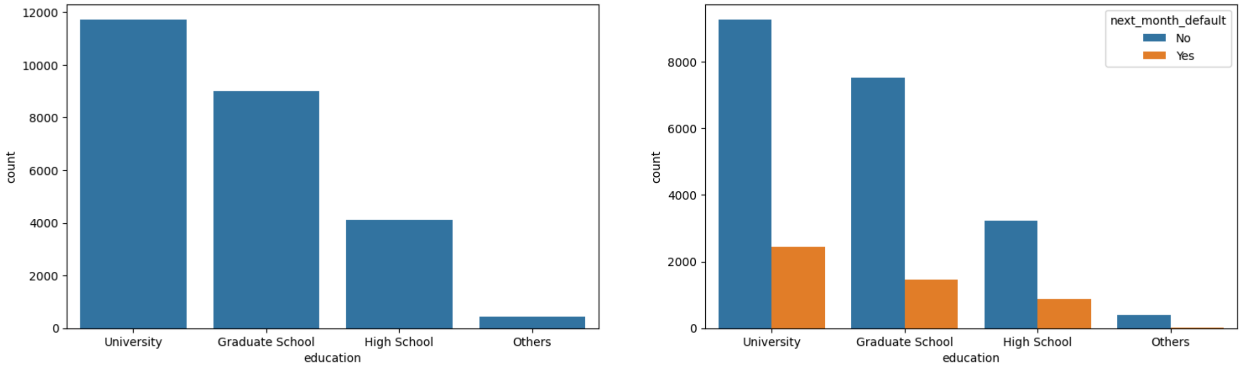
Here, we can see that it has very less null values and only for the ‘age’ column, we can impute them by analysing its graph.

2. next_month_default column



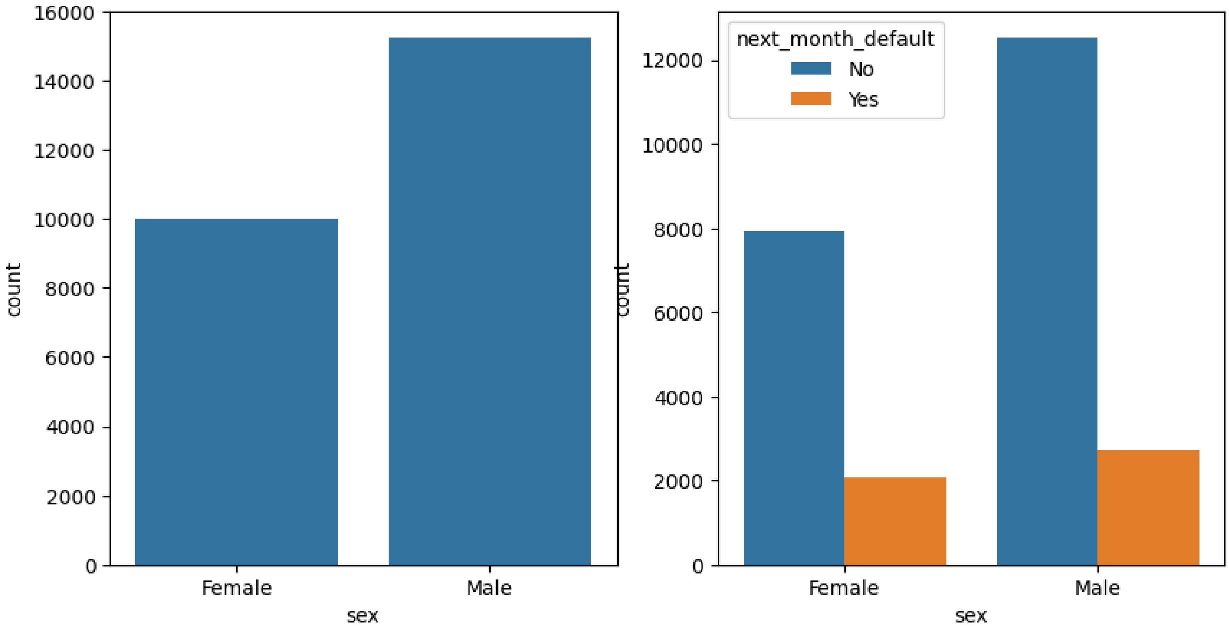
Here, it is clear that the dataset is skewed, and should be balanced before training any model for better results.

3. Education column (Categorical column)



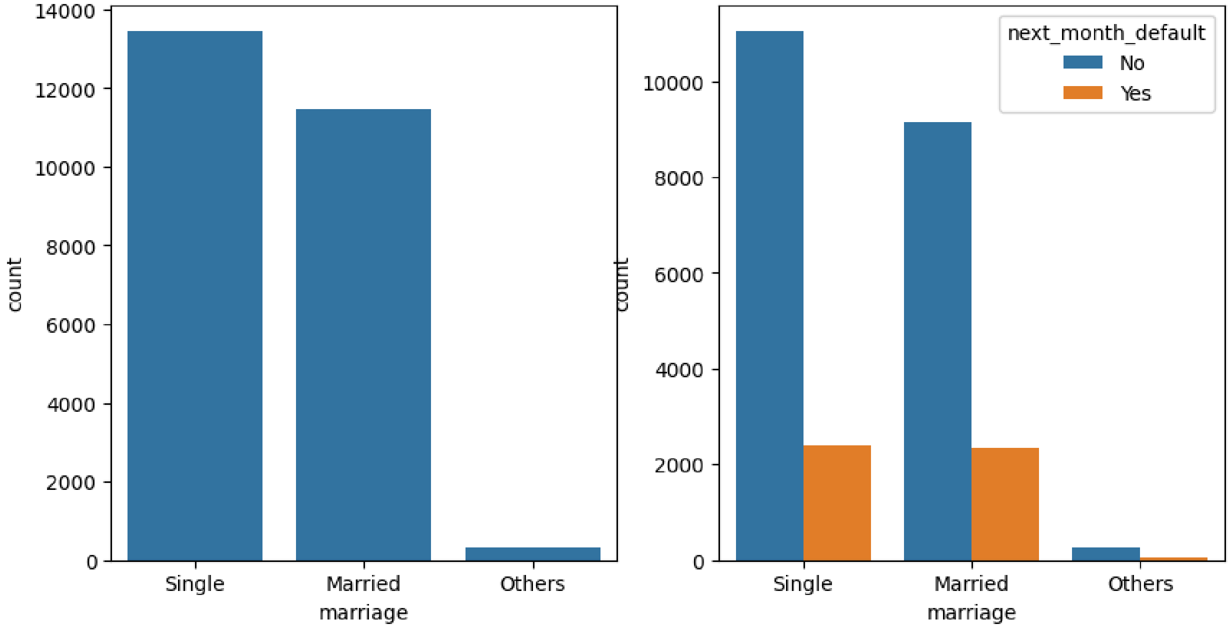
The proportion of defaulters among Graduate school students is quite high, and for ‘Others’ it is negligible, removing those would not cause any harm.

4. Sex column



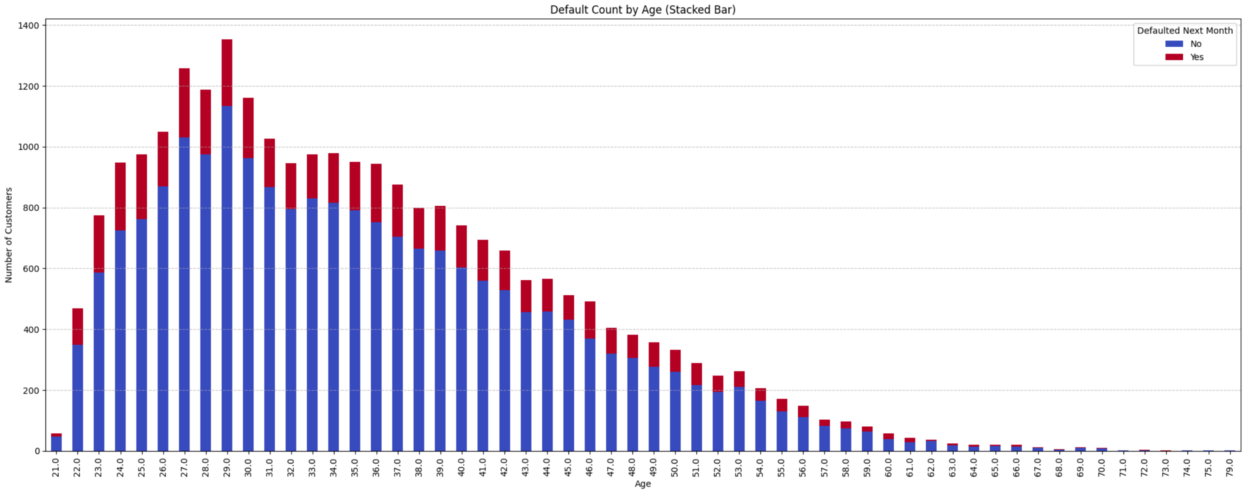
This is also a categorical column, suggesting using one hot encoding for it

5. ‘Marriage’ column

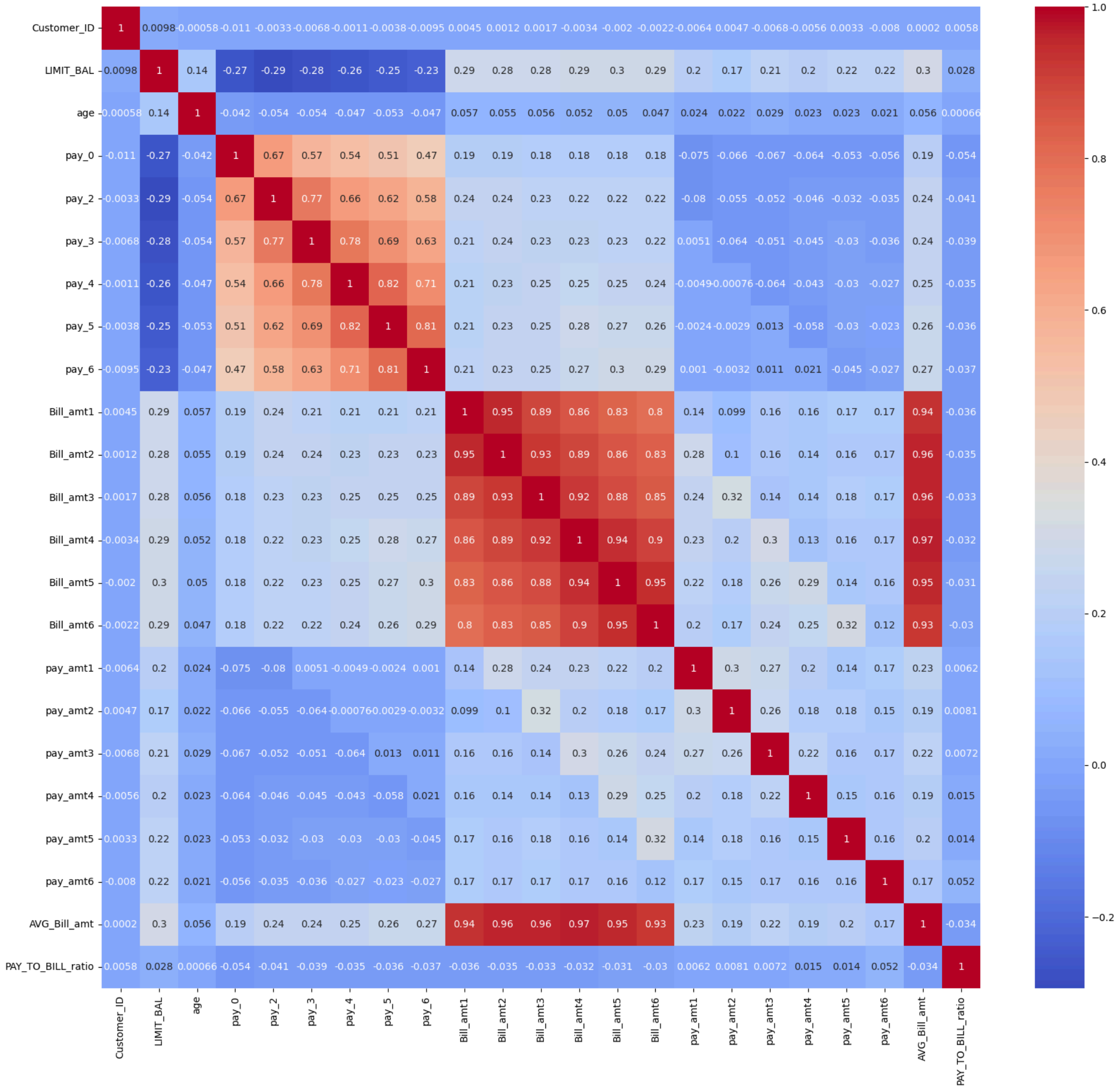


Both single and married have good proportion of defaulters.

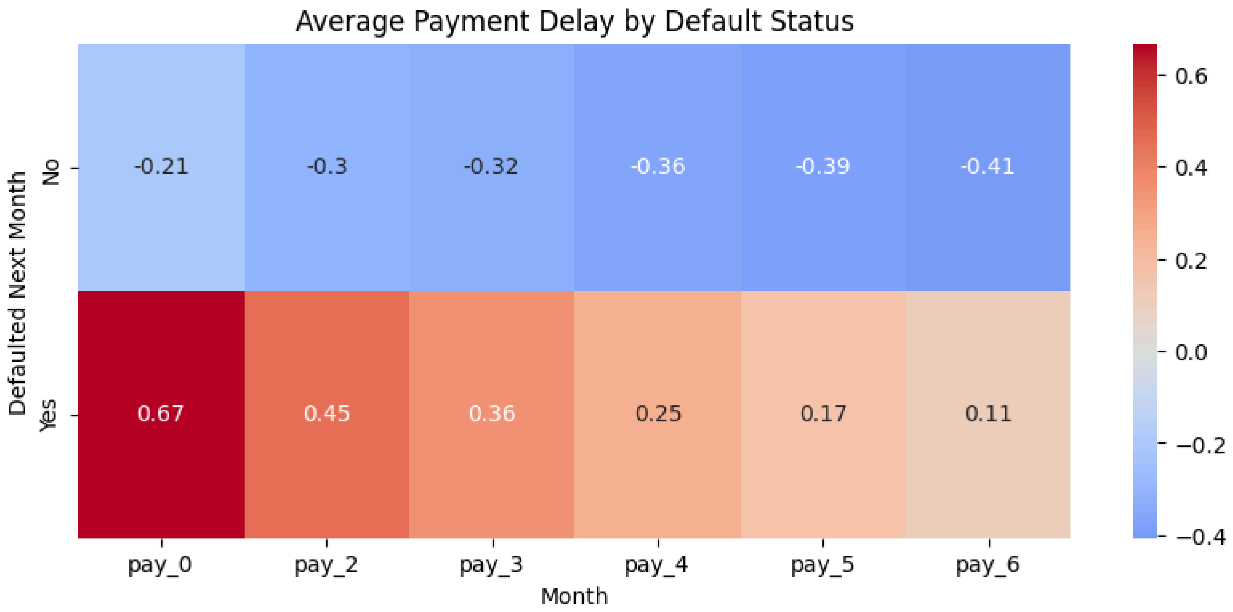
6. ‘Age’ column



7. Correlation matrix

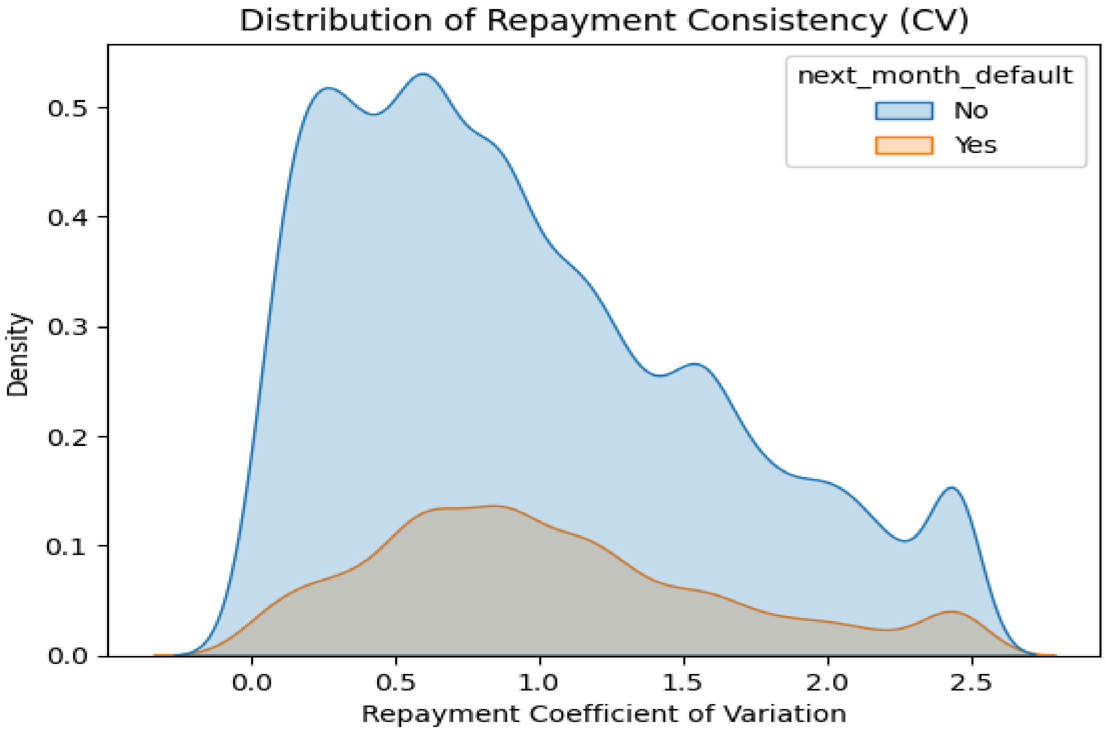


## 8. Average payment delay by default status



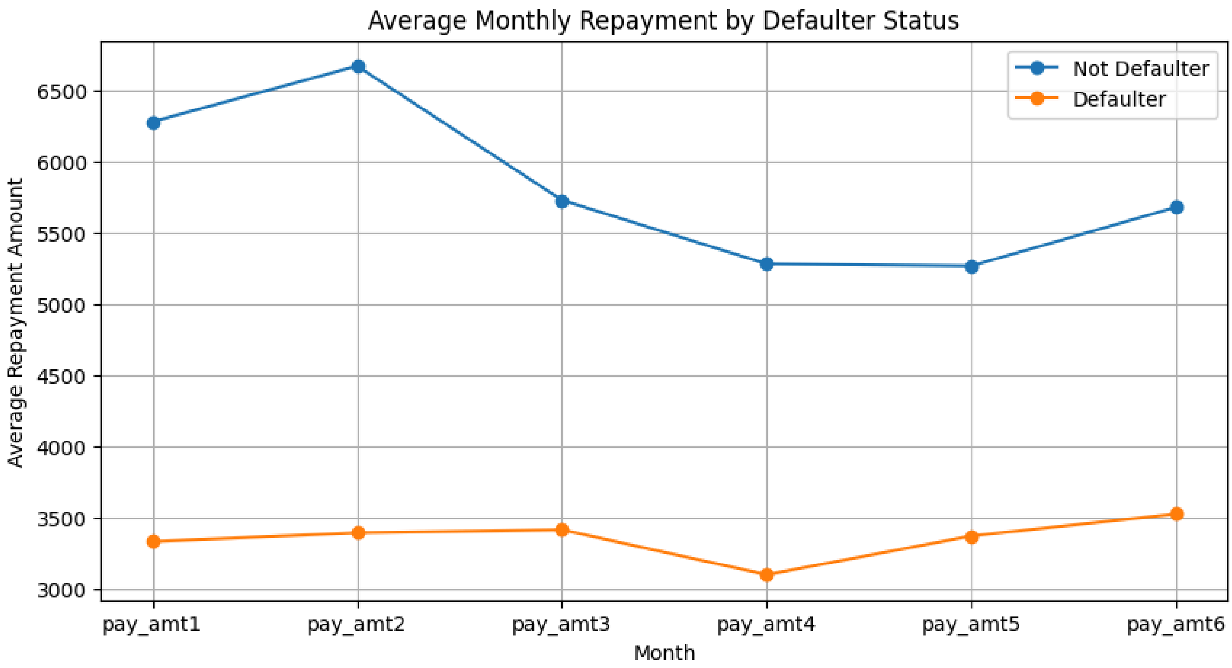
This plot is quite obvious as it shows the decreasing probability of defaulting in ongoing months.

9. Repayment coefficient of variation



A **higher CV** implies **more inconsistent repayment behavior**, while a **lower CV** suggests stable, regular payments.

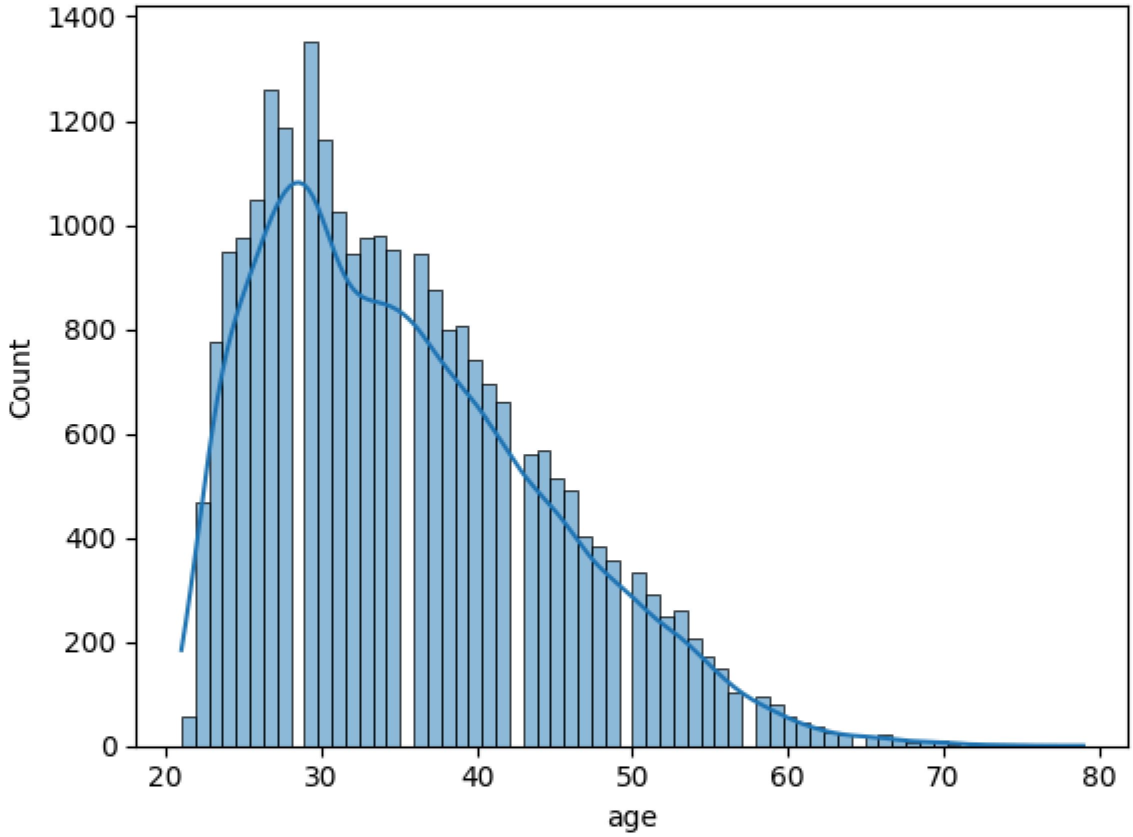
10. Average monthly repayment



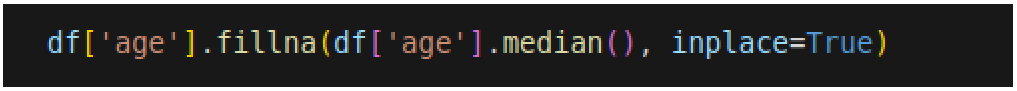
This plot is also obvious as the average repayment amount is very less for defaulters as compared to non defaulters.

## Data Preprocessing

- Filling NaN values for ‘age’ column

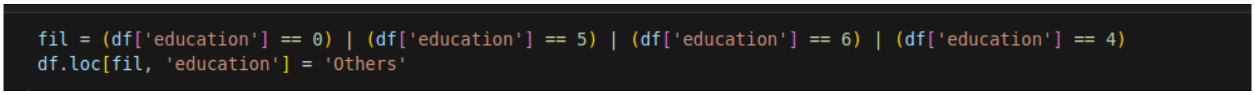


Since it has kind of right skewed ,we can follow median imputation. (Tried filling with correlation to other columns but this works best)



```
df['age'].fillna(df['age'].median(), inplace=True)
```

- Handling different values in ‘education’ column

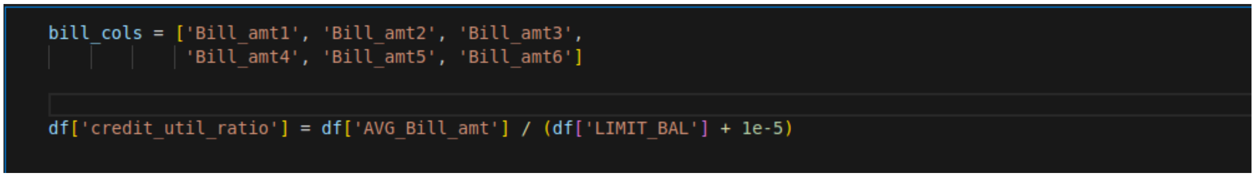


```
fil = (df['education'] == 0) | (df['education'] == 5) | (df['education'] == 6) | (df['education'] == 4)
df.loc[fil, 'education'] = 'Others'
```

All values with 0,5,6,4 clubbed into ‘Others’.

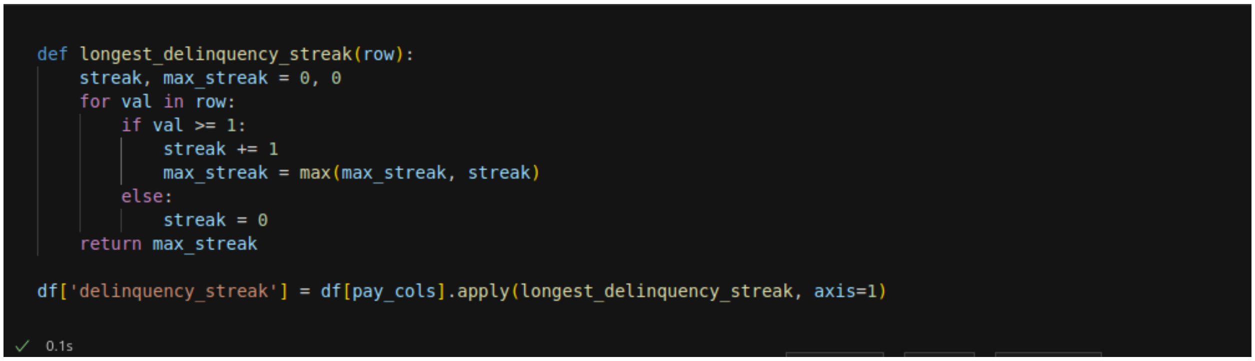
## Feature Engineering

- Credit_util_ratio (The `credit_util_ratio` feature represents the **average proportion of a customer's credit limit used**, calculated by dividing their **average bill amount** over six months by their **credit limit**.)



```
bill_cols = ['Bill_amt1', 'Bill_amt2', 'Bill_amt3',
             'Bill_amt4', 'Bill_amt5', 'Bill_amt6']

df['credit_util_ratio'] = df['AVG_Bill_amt'] / (df['LIMIT_BAL'] + 1e-5)
```

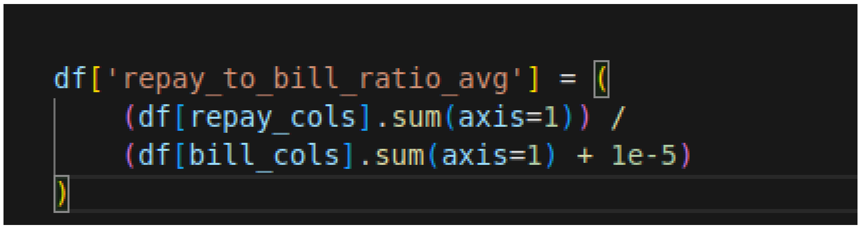
- Deliquency_streak (The `delinquency_streak` feature captures the **longest consecutive number of months** a customer was **1 or more months late on payments**, indicating persistent delinquency behavior.)



```
def longest_delinquency_streak(row):
    streak, max_streak = 0, 0
    for val in row:
        if val >= 1:
            streak += 1
            max_streak = max(max_streak, streak)
        else:
            streak = 0
    return max_streak

df['delinquency_streak'] = df[pay_cols].apply(longest_delinquency_streak, axis=1)
```

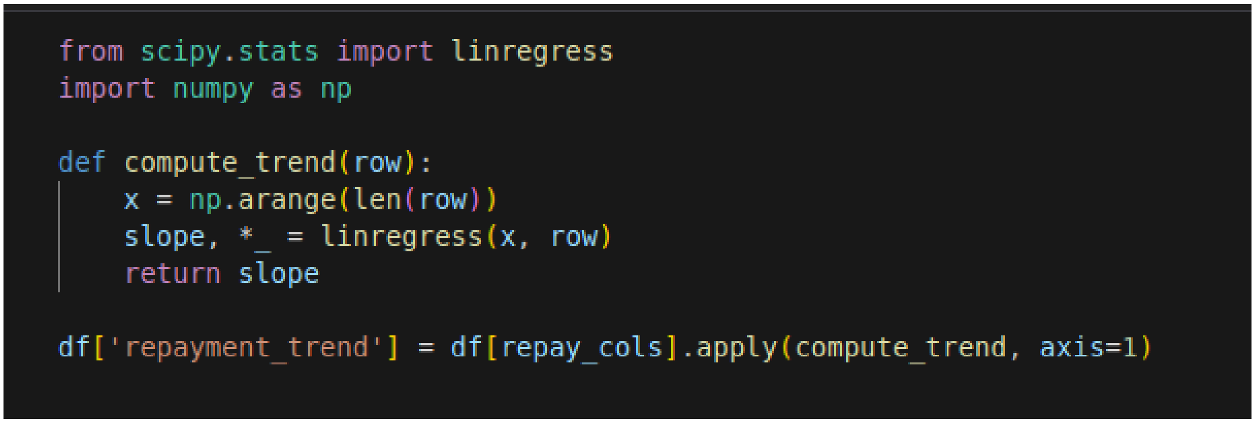
✓ 0.1s

- Repay_to_bill_ratio_avg (The `repay_to_bill_ratio_avg` feature represents the **average proportion of total repayments to total billed amounts** over 6 months, indicating how consistently a customer pays off their bills.)



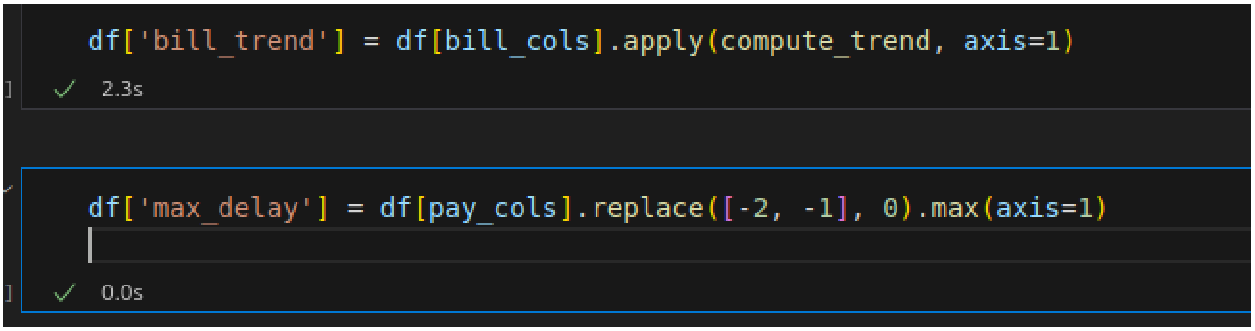
```
df['repay_to_bill_ratio_avg'] = (
    (df[repay_cols].sum(axis=1)) /
    (df[bill_cols].sum(axis=1) + 1e-5)
)
```

- Repayment_trend (The `repayment_trend` feature captures the **overall increasing or decreasing trend** in a customer's repayments over time by calculating the **slope of a linear regression line** through their monthly repayment amounts.)



```
from scipy.stats import linregress
import numpy as np

def compute_trend(row):
    x = np.arange(len(row))
    slope, *_ = linregress(x, row)
    return slope

df['repayment_trend'] = df[repay_cols].apply(compute_trend, axis=1)
```

- max_delay and bill_trend (`bill_trend`: Captures the **direction and rate of change** in a customer's bill amounts over time using the **slope of a linear trend**.
`max_delay`: Represents the **maximum payment delay** experienced by a customer, ignoring early or on-time payments (i.e., negative values treated as zero).



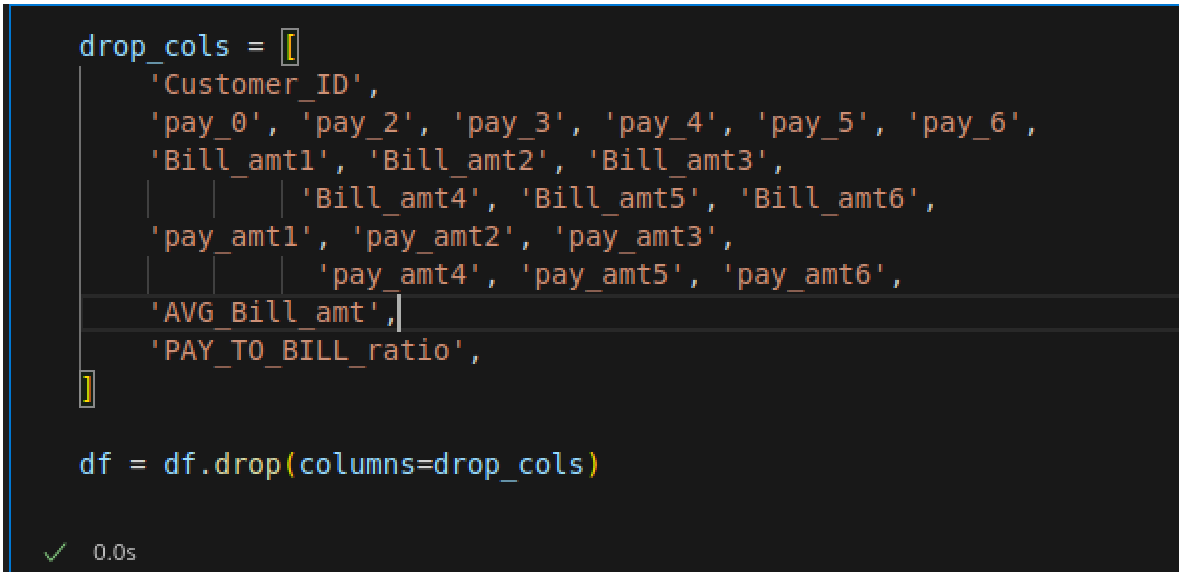
```
df['bill_trend'] = df[bill_cols].apply(compute_trend, axis=1)
```
✓ 2.3s

```
df['max_delay'] = df[pay_cols].replace([-2, -1], 0).max(axis=1)
```
✓ 0.0s

## Feature selection

After performing feature engineering, I analyzed the **correlation matrix**, which included both the original and newly created features. This analysis revealed that several newly engineered features had **strong correlations** with some of the original features they were derived from.

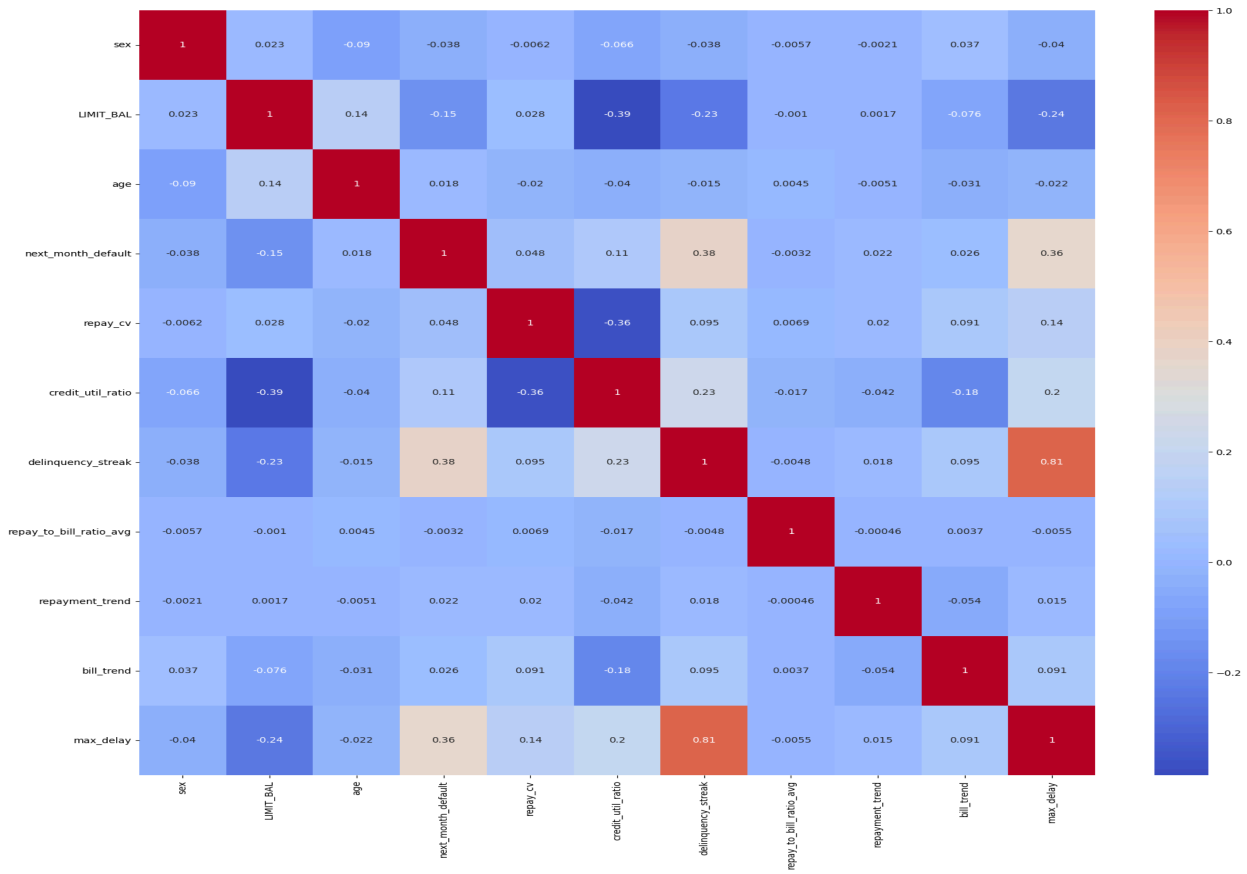
To avoid **multicollinearity** and **redundant information** in the model, I decided to **drop the original features** that were already effectively represented within the engineered ones. This not only helps in reducing dimensionality but also improves the model's **interpretability** and **generalization** by focusing on more meaningful predictors.



```
drop_cols = [
    'Customer_ID',
    'pay_0', 'pay_2', 'pay_3', 'pay_4', 'pay_5', 'pay_6',
    'Bill_amt1', 'Bill_amt2', 'Bill_amt3',
             'Bill_amt4', 'Bill_amt5', 'Bill_amt6',
    'pay_amt1', 'pay_amt2', 'pay_amt3',
              'pay_amt4', 'pay_amt5', 'pay_amt6',
    'AVG_Bill_amt',
    'PAY_TO_BILL_ratio',
]

df = df.drop(columns=drop_cols)
```

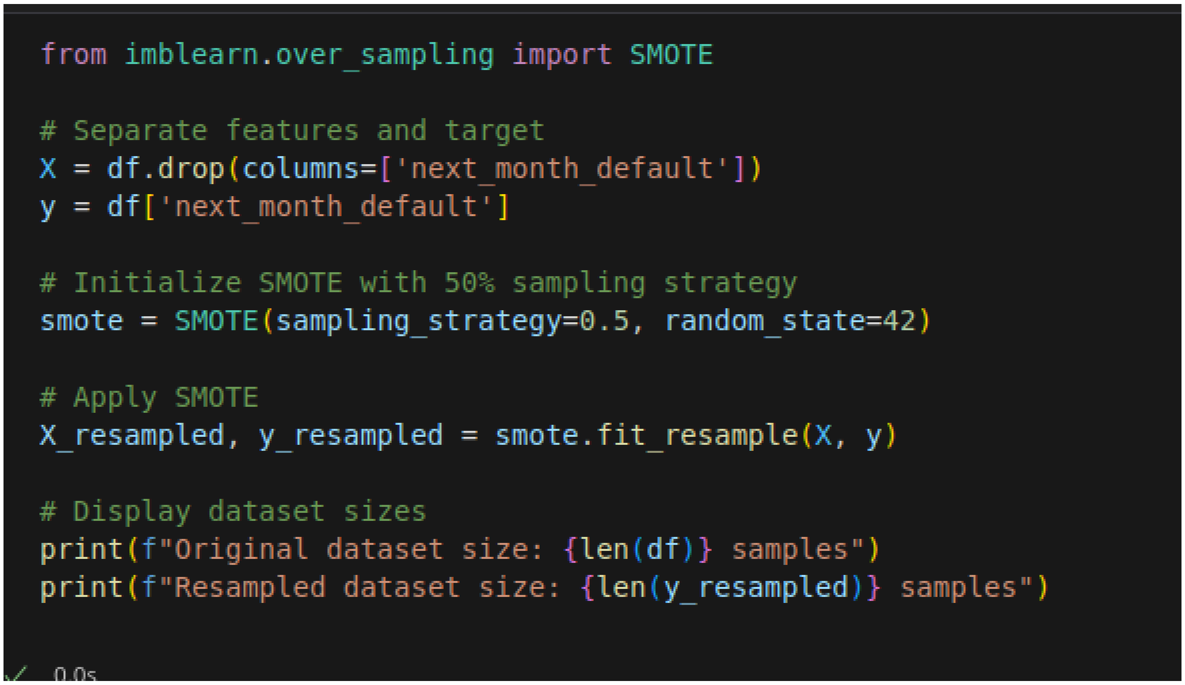
✓ 0.0s

Correlation matrix after dropping-



## Model Training-

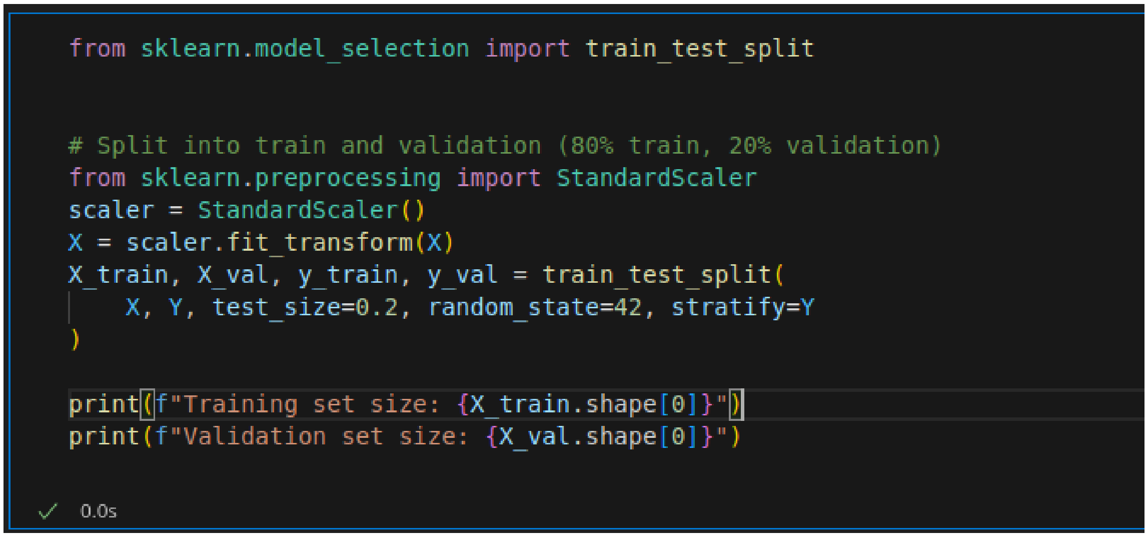
- Applying SMOTE to handle class imbalance-



```python
from imblearn.over_sampling import SMOTE

# Separate features and target
X = df.drop(columns=['next_month_default'])
y = df['next_month_default']

# Initialize SMOTE with 50% sampling strategy
smote = SMOTE(sampling_strategy=0.5, random_state=42)

# Apply SMOTE
X_resampled, y_resampled = smote.fit_resample(X, y)

# Display dataset sizes
print(f"Original dataset size: {len(df)} samples")
print(f"Resampled dataset size: {len(y_resampled)} samples")
```

✓ 0.0s

- 80-20 split for training and validation

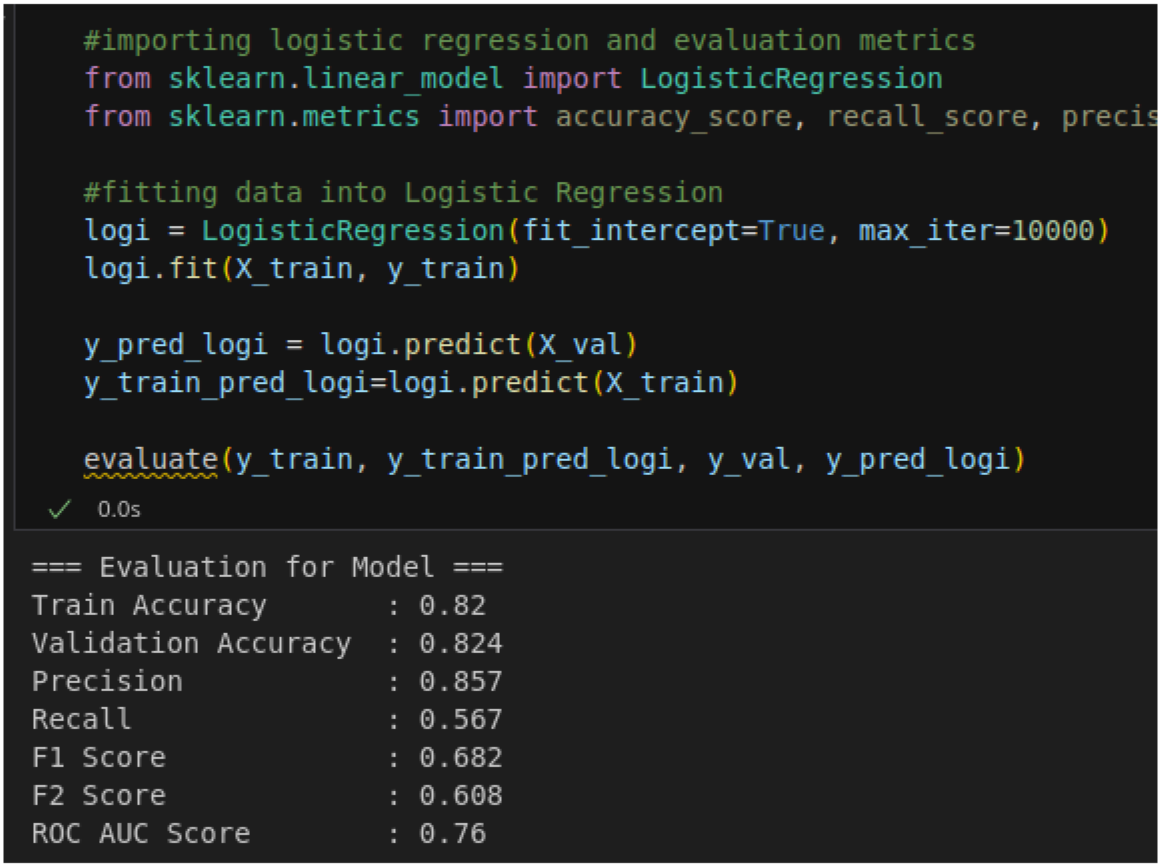


```python
from sklearn.model_selection import train_test_split


# Split into train and validation (80% train, 20% validation)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_val, y_train, y_val = train_test_split(
    X, Y, test_size=0.2, random_state=42, stratify=Y
)

print(f"Training set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
```

✓ 0.0s

- Testing different models-

1. Logistic Regression -

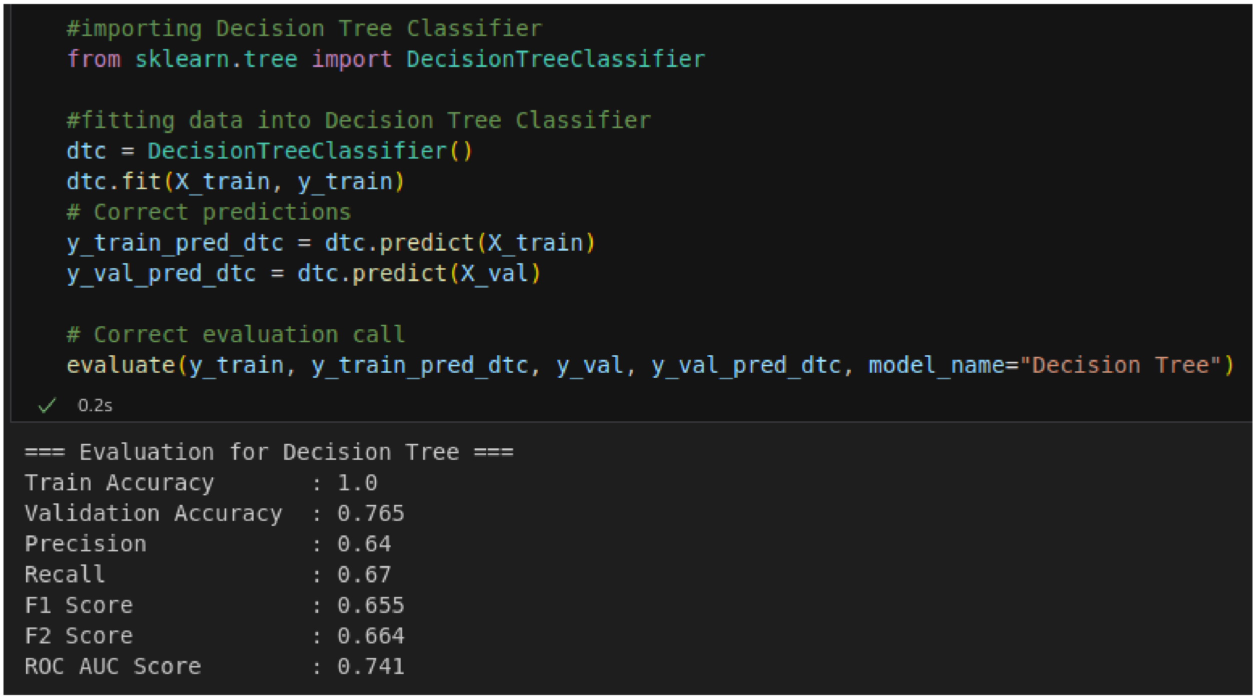


```python
#importing logistic regression and evaluation metrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precis

#fitting data into Logistic Regression
logi = LogisticRegression(fit_intercept=True, max_iter=10000)
logi.fit(X_train, y_train)

y_pred_logi = logi.predict(X_val)
y_train_pred_logi=logi.predict(X_train)

evaluate(y_train, y_train_pred_logi, y_val, y_pred_logi)
```

```
=== Evaluation for Model ===
Train Accuracy       : 0.82
Validation Accuracy  : 0.824
Precision            : 0.857
Recall               : 0.567
F1 Score             : 0.682
F2 Score             : 0.608
ROC AUC Score        : 0.76
```

F2 Score- 0.608

2. Decision Trees-

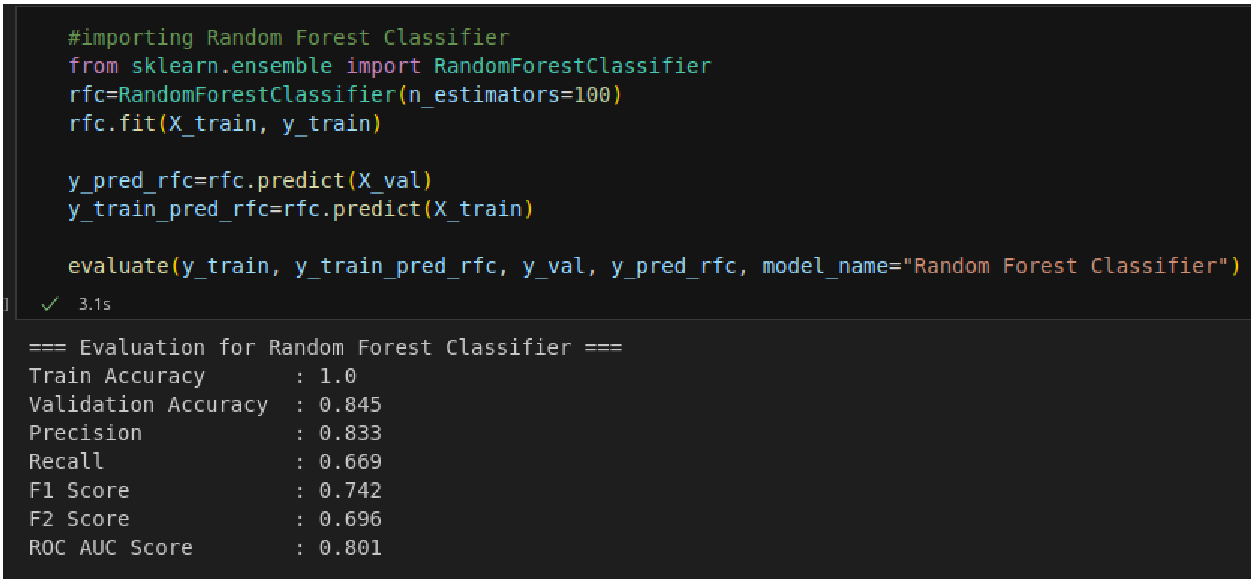


```
#importing Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

#fitting data into Decision Tree Classifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
# Correct predictions
y_train_pred_dtc = dtc.predict(X_train)
y_val_pred_dtc = dtc.predict(X_val)

# Correct evaluation call
evaluate(y_train, y_train_pred_dtc, y_val, y_val_pred_dtc, model_name="Decision Tree")
```

```
=== Evaluation for Decision Tree ===
Train Accuracy       : 1.0
Validation Accuracy  : 0.765
Precision            : 0.64
Recall               : 0.67
F1 Score             : 0.655
F2 Score             : 0.664
ROC AUC Score        : 0.741
```

F2 score- 0.664

3. Random forest classifier



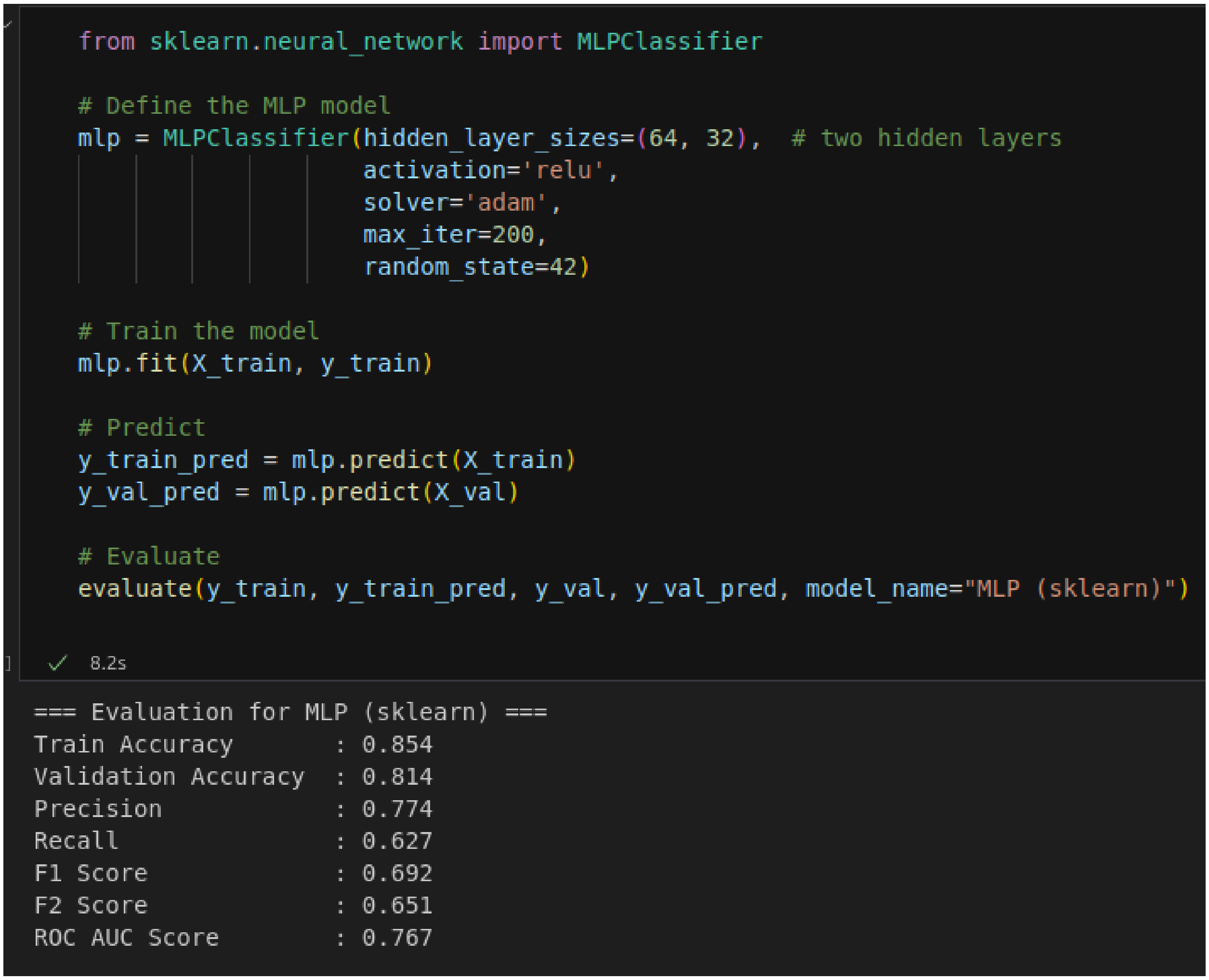
```
#importing Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)

y_pred_rfc=rfc.predict(X_val)
y_train_pred_rfc=rfc.predict(X_train)

evaluate(y_train, y_train_pred_rfc, y_val, y_pred_rfc, model_name="Random Forest Classifier")
```

```
=== Evaluation for Random Forest Classifier ===
Train Accuracy       : 1.0
Validation Accuracy  : 0.845
Precision            : 0.833
Recall               : 0.669
F1 Score             : 0.742
F2 Score             : 0.696
ROC AUC Score        : 0.801
```

F2 score- 0.696

4. Multi Layer perceptrons (MLP)

5.



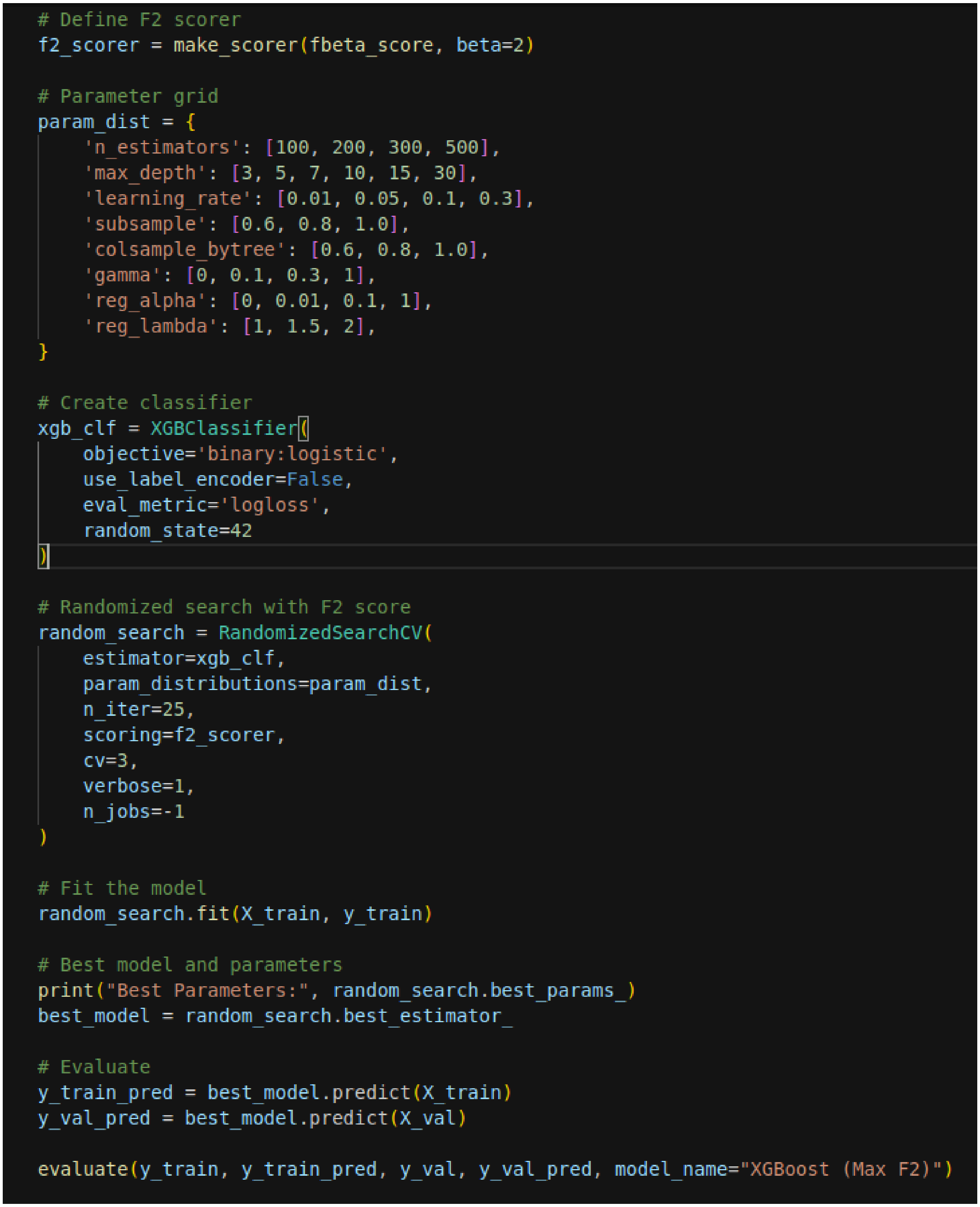
```
from sklearn.neural_network import MLPClassifier

# Define the MLP model
mlp = MLPClassifier(hidden_layer_sizes=(64, 32),  # two hidden layers
                    activation='relu',
                    solver='adam',
                    max_iter=200,
                    random_state=42)

# Train the model
mlp.fit(X_train, y_train)

# Predict
y_train_pred = mlp.predict(X_train)
y_val_pred = mlp.predict(X_val)

# Evaluate
evaluate(y_train, y_train_pred, y_val, y_val_pred, model_name="MLP (sklearn)")
```

✓ 8.2s

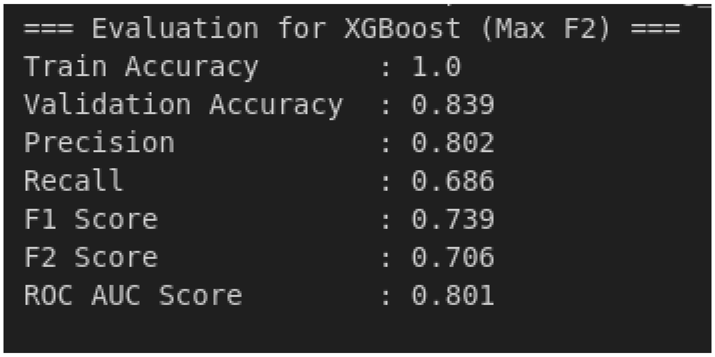
```
=== Evaluation for MLP (sklearn) ===
Train Accuracy       : 0.854
Validation Accuracy  : 0.814
Precision            : 0.774
Recall               : 0.627
F1 Score             : 0.692
F2 Score             : 0.651
ROC AUC Score        : 0.767
```

F2 score- 0.651

6. XGBoost (with Grid search)



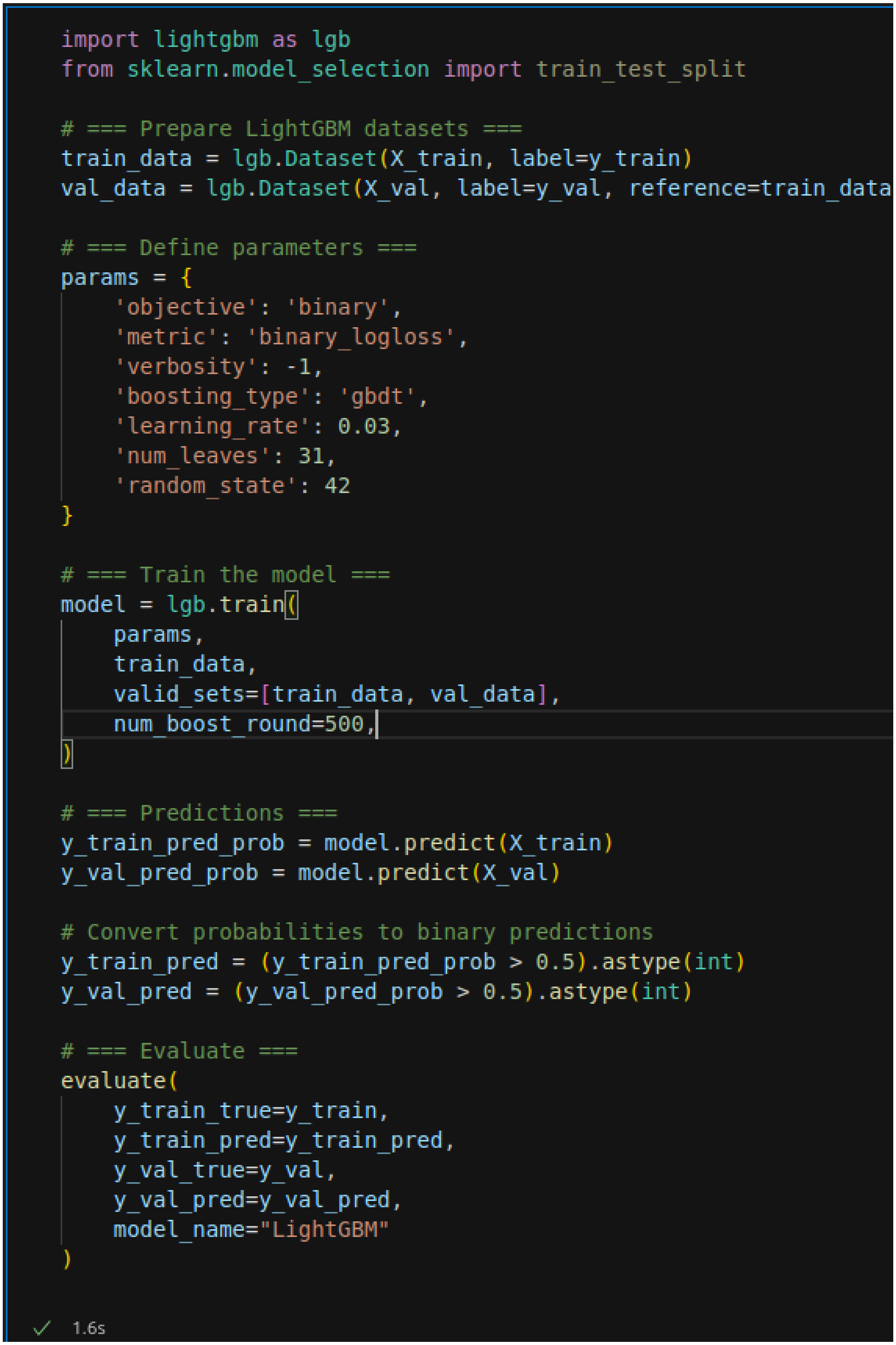
```
# Define F2 scorer
f2_scorer = make_scorer(fbeta_score, beta=2)

# Parameter grid
param_dist = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [3, 5, 7, 10, 15, 30],
    'learning_rate': [0.01, 0.05, 0.1, 0.3],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.3, 1],
    'reg_alpha': [0, 0.01, 0.1, 1],
    'reg_lambda': [1, 1.5, 2],
}

# Create classifier
xgb_clf = XGBClassifier(
    objective='binary:logistic',
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)

# Randomized search with F2 score
random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist,
    n_iter=25,
    scoring=f2_scorer,
    cv=3,
    verbose=1,
    n_jobs=-1
)

# Fit the model
random_search.fit(X_train, y_train)

# Best model and parameters
print("Best Parameters:", random_search.best_params_)
best_model = random_search.best_estimator_

# Evaluate
y_train_pred = best_model.predict(X_train)
y_val_pred = best_model.predict(X_val)

evaluate(y_train, y_train_pred, y_val, y_val_pred, model_name="XGBoost (Max F2)")
```



```
=== Evaluation for XGBoost (Max F2) ===
Train Accuracy       : 1.0
Validation Accuracy  : 0.839
Precision            : 0.802
Recall               : 0.686
F1 Score             : 0.739
F2 Score             : 0.706
ROC AUC Score        : 0.801
```

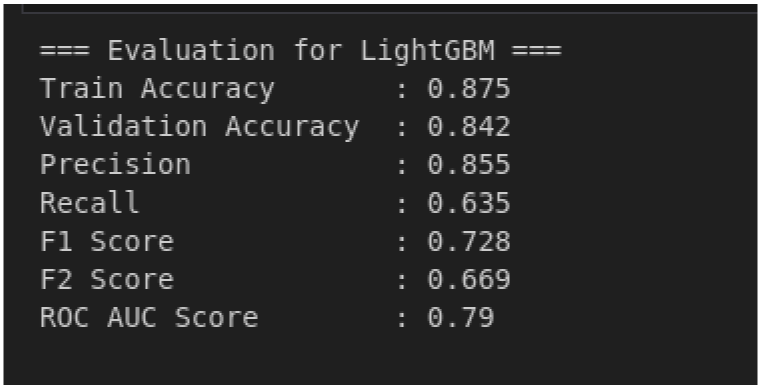
F2 score- 0.706

7. Light GBM



```python
import lightgbm as lgb
from sklearn.model_selection import train_test_split

# === Prepare LightGBM datasets ===
train_data = lgb.Dataset(X_train, label=y_train)
val_data = lgb.Dataset(X_val, label=y_val, reference=train_data

# === Define parameters ===
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'verbosity': -1,
    'boosting_type': 'gbdt',
    'learning_rate': 0.03,
    'num_leaves': 31,
    'random_state': 42
}

# === Train the model ===
model = lgb.train(
    params,
    train_data,
    valid_sets=[train_data, val_data],
    num_boost_round=500,
)

# === Predictions ===
y_train_pred_prob = model.predict(X_train)
y_val_pred_prob = model.predict(X_val)

# Convert probabilities to binary predictions
y_train_pred = (y_train_pred_prob > 0.5).astype(int)
y_val_pred = (y_val_pred_prob > 0.5).astype(int)

# === Evaluate ===
evaluate(
    y_train_true=y_train,
    y_train_pred=y_train_pred,
    y_val_true=y_val,
    y_val_pred=y_val_pred,
    model_name="LightGBM"
)
```

✓ 1.6s

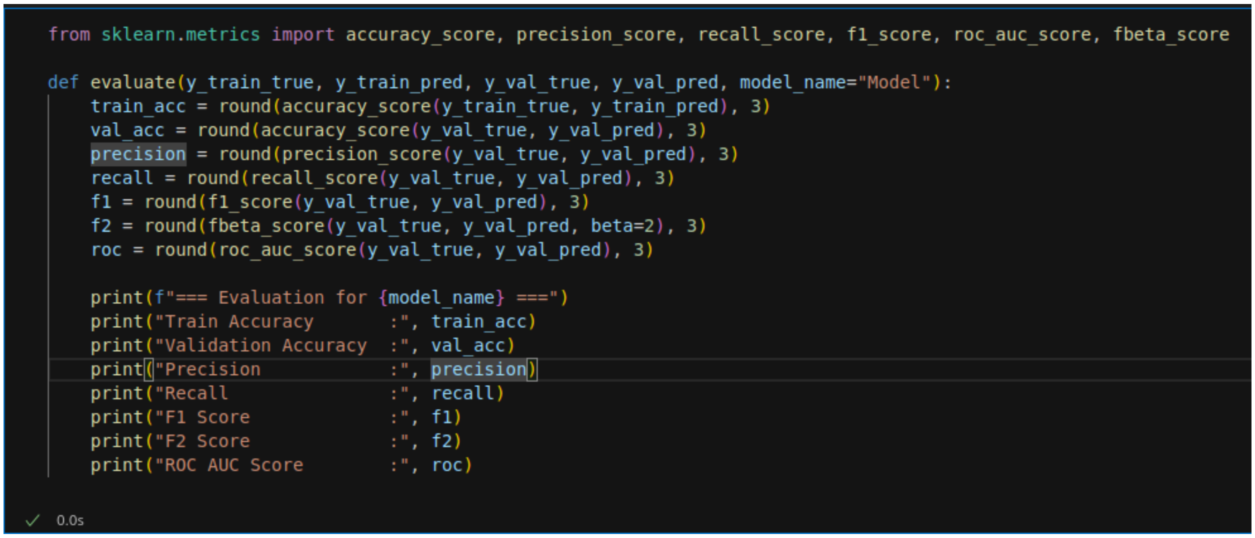


```
=== Evaluation for LightGBM ===
Train Accuracy       : 0.875
Validation Accuracy  : 0.842
Precision            : 0.855
Recall               : 0.635
F1 Score             : 0.728
F2 Score             : 0.669
ROC AUC Score        : 0.79
```

F2 score- 0.669

To address the class imbalance in the dataset, I applied **SMOTE** with a **sampling strategy of 50%**, meaning the minority class (defaulters) was increased to 50% of the majority class. This approach was chosen deliberately to avoid excessive oversampling, which can lead to **overfitting** by introducing synthetic patterns that may not generalize well to unseen data.

All models were trained on the resampled dataset, and their performance was evaluated using appropriate metrics — including **precision, recall, F1-score, and AUC-ROC** — to ensure a balanced assessment of predictive ability, especially given the class imbalance context.



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, fbeta_score

def evaluate(y_train_true, y_train_pred, y_val_true, y_val_pred, model_name="Model"):
    train_acc = round(accuracy_score(y_train_true, y_train_pred), 3)
    val_acc = round(accuracy_score(y_val_true, y_val_pred), 3)
    precision = round(precision_score(y_val_true, y_val_pred), 3)
    recall = round(recall_score(y_val_true, y_val_pred), 3)
    f1 = round(f1_score(y_val_true, y_val_pred), 3)
    f2 = round(fbeta_score(y_val_true, y_val_pred, beta=2), 3)
    roc = round(roc_auc_score(y_val_true, y_val_pred), 3)

    print(f"=== Evaluation for {model_name} ===")
    print("Train Accuracy       :", train_acc)
    print("Validation Accuracy  :", val_acc)
    print("Precision            :", precision)
    print("Recall               :", recall)
    print("F1 Score             :", f1)
    print("F2 Score             :", f2)
    print("ROC AUC Score        :", roc)
```

✓ 0.0s

This ‘evaluate’ function was used to evaluate the models. Observing we get the best F2 Score from **XGBoost** model which was **0.706.**