Name-Minaksi Yadav

Credit Card Default prediction using Machine Learning Techniques

REPORT:

Overview

The primary objective is to build a classification model that predicts whether a credit card customer is likely to default in the upcoming billing cycle.

To achieve this, anonymized behavioral data from over 30,000 customers has been analyzed. The dataset includes a labeled target variable, `default.payment.next.month`, which indicates default status in the next month.

The goal is not only to accurately identify potential defaulters early but also to develop a model that is interpretable from a financial standpoint. Such a model enables the bank to take proactive measures—like adjusting credit limits, issuing alerts, or prioritizing high-risk cases—to manage credit exposure more effectively and reduce potential losses.

Trajectory Of the Project

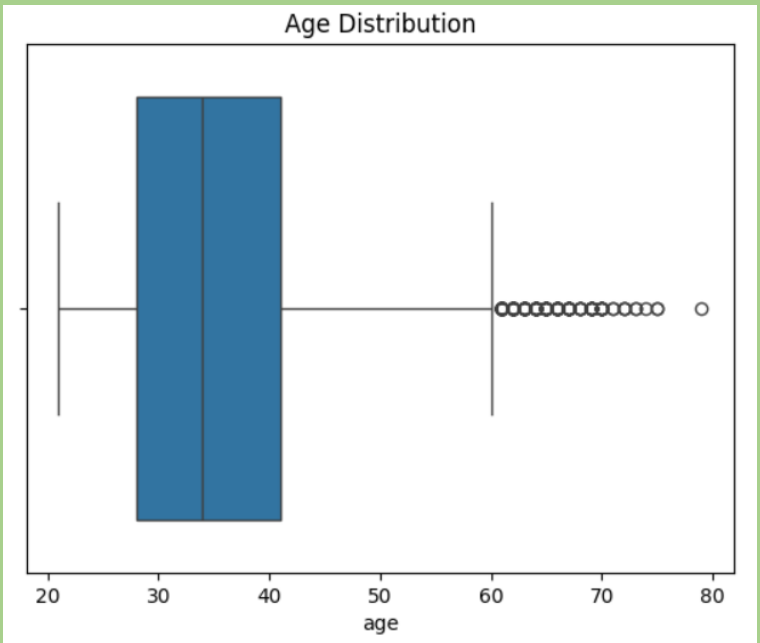
Step-1

First we Start with analysing the Data we are given

So upon checking for missing values we found that age is the only column with missing values

So we plotted the box plot and observed most people having credit card are between 30s and 40s and age data also Have some outliers so we will replace the missing values with the median

Here is the box Plot



So missing values are done now and dataset have no missing values left

Step-2

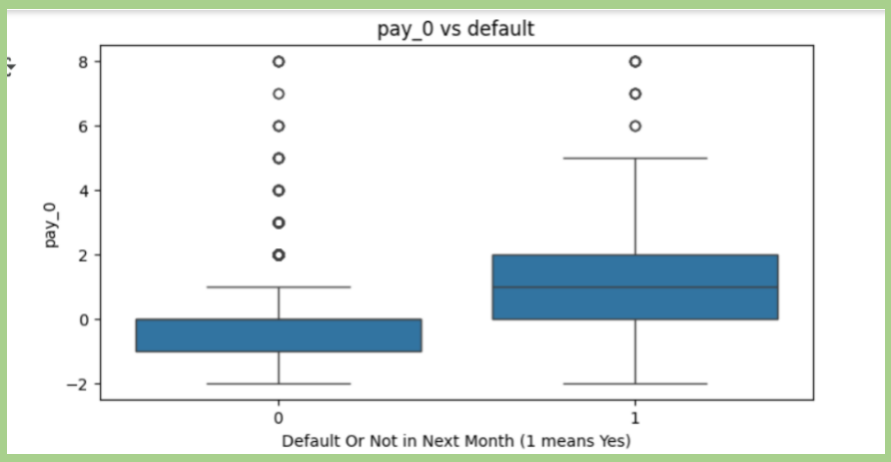
we will now see all columns vs default status 1 by 1

1. pay_m ->

PAY_0 represents the payment status in the most recent month (Month 0), PAY_2 represents the payment status one month before that, and so on.
Values:

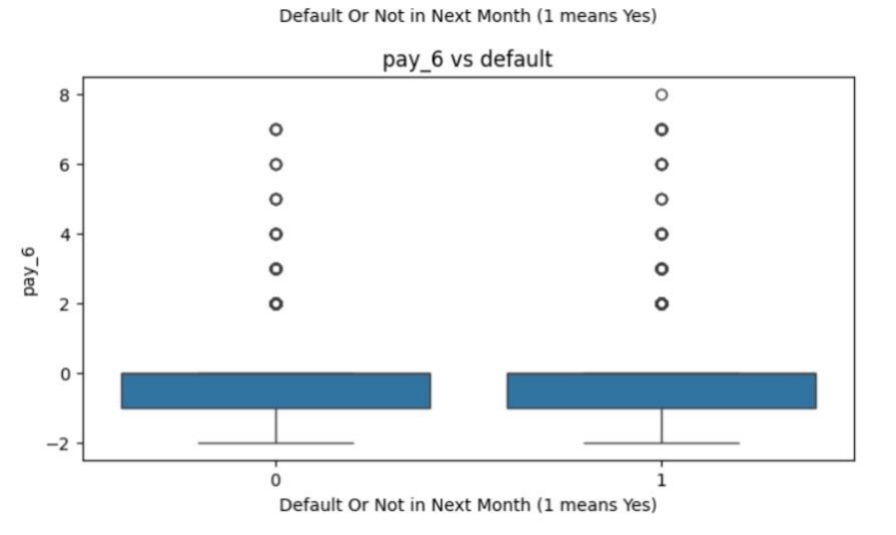
- -2 = No credit consumption (no bill) in month *m−1*
- -1 = Bill generated and fully paid in month *m−1* (same month payment)
- 0= Partial or minimum payment made (revolving credit)

≥1 = Payment delayed by that many months (1 = 1 month overdue, etc.)
when I plotted boxplot of all of them I just observed that guys are going to default in next month are either making partial payment or delaying that by some months as seen below in Graphs



And the same graph for pay_2,pay_3,pay_4

But for pay_5,pay_6



Graph is like this for pay_5,pay_6 Its obvious back in the past they must be paying bills fully or partially

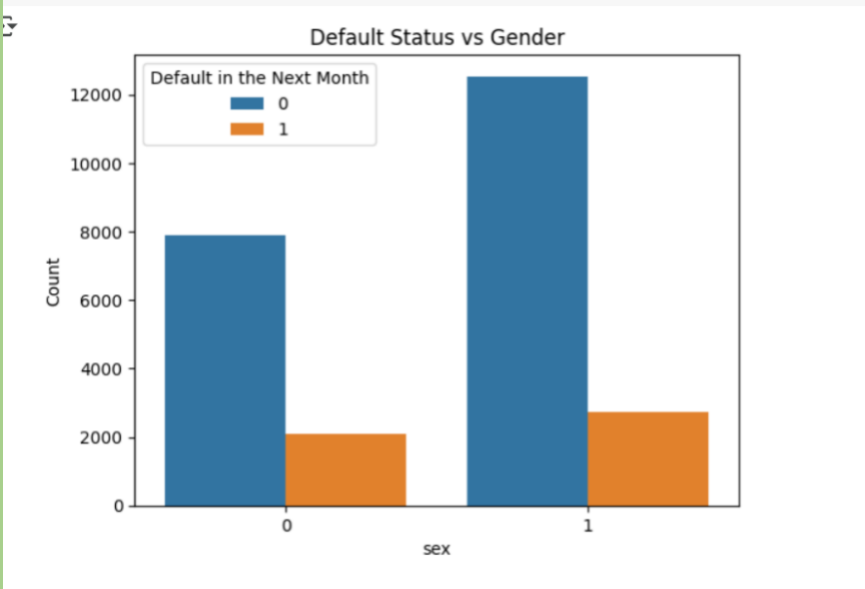
2. Age

Boxplot for age vs defaulter

Cannot really say something as both are lying in the same age group

3. Sex

Boxplot for sex vs defaulter(0->Female 1->Male)



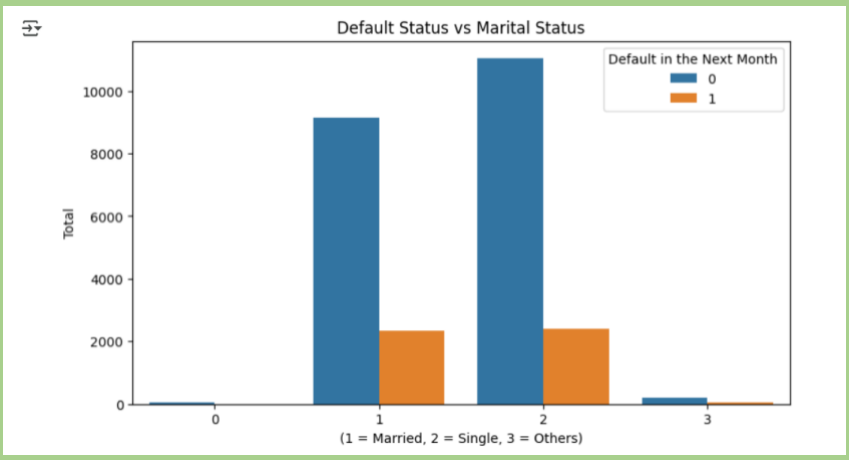
Proportionality of being default is more in Females than Males

```
[ ]  # The Proportionality of default in male is slightly
     train_df.groupby('sex')['next_month_default'].mean()
```

sex	next_month_default
Female	0.208604
Male	0.178468

dtype: float64

4. Marriage ->While Plotting boxplot for marriage I observed that There are some extra values having values 0



I combined them with others i.e. 3

```
#As we can see that 0 is nearly negligible so we will combine that with others
train_df['marriage'].value_counts()
```

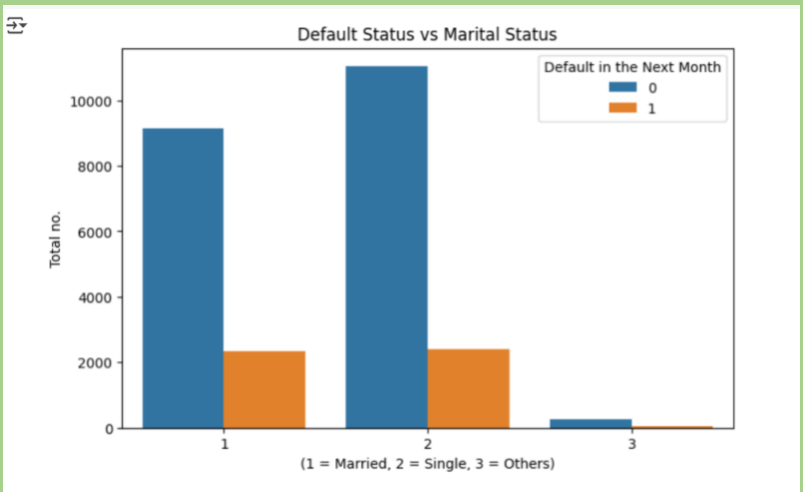
marriage	count
2	13441
1	11480
3	273
0	53

dtype: int64

Code for combining:

```
train_df['marriage'] = train_df['marriage'].replace({0: 3})
```

Boxplot after Combining



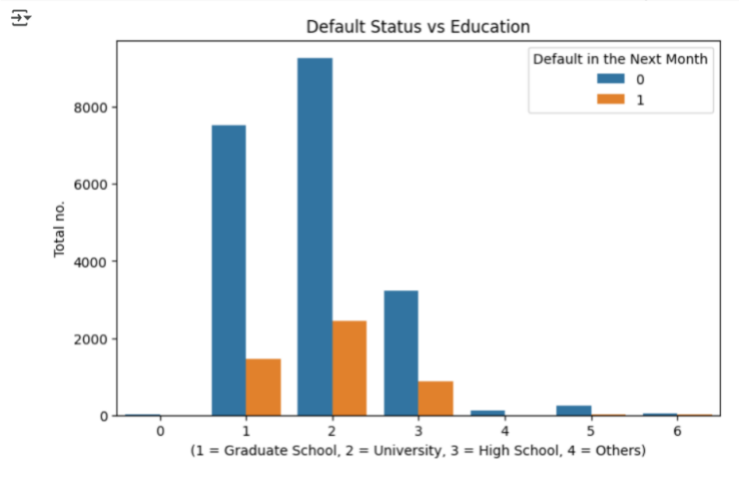
Singles are having proportionally slightly more chance to default

```
train_df.groupby('marriage')['next_month_default'].mean()
```

marriage	next_month_default
1	0.203746
2	0.178856
3	0.196319

5. Education:-

Boxplot for education vs Defaulters

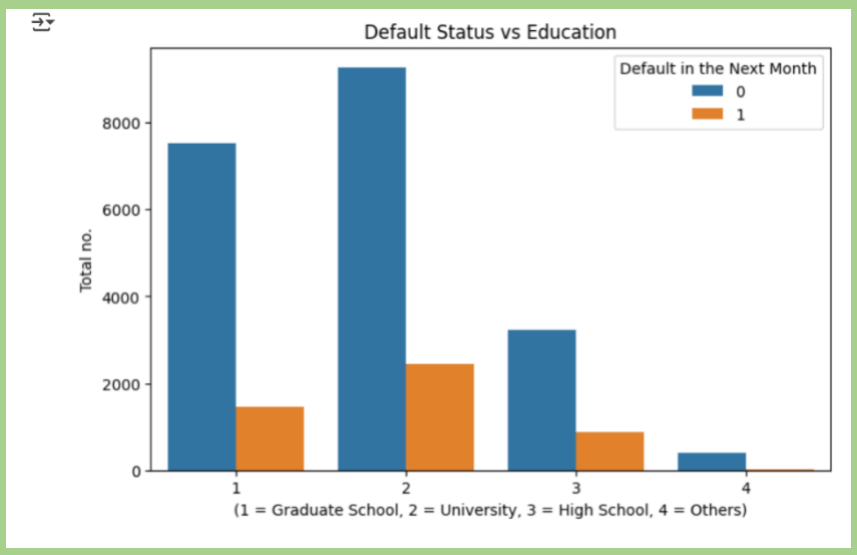


Here also values other than 1 2 3 4 are observed so we will combine 0,5,6 with 4 i.e others

Code For Combining:-

```
[ ]  train_df['education'] = train_df['education'].replace({0: 4, 5: 4, 6: 4})
```

Boxplot After Combing:



```
[ ] train_df.groupby('education')['next_month_default'].mean()
```

education	next_month_default
1	0.161771
2	0.209098
3	0.213123
4	0.060890

dtype: float64

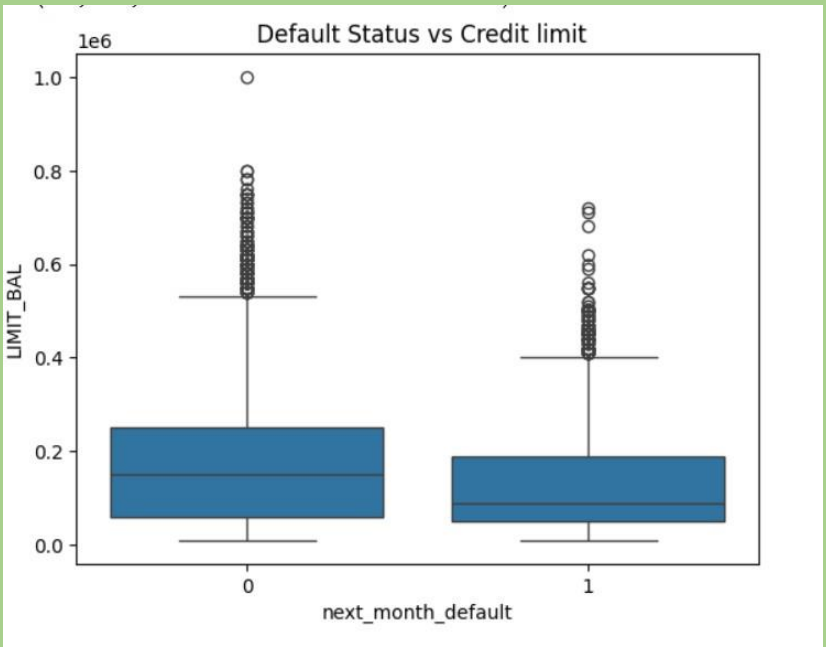
```
[ ] #So Here Values other than 1 2 3 4 are included in others(4)
    #Also default rate high in 2(University) and 3(High School) and slightly low in 1(University)
```

So default rate is slightly high in University and High school as compared to University

6. Limit_Bal:

So upon plotting the box plot I observed that More limit gives less of defaulting and vice versa

Boxplot for Limit_bal vs defaulter



So we can say with this Our EDA part is done and now will be moving towards the Feature Engineering part of this Project

So Step-2 is done and now

Step-3

Feature Engineering :-

```
[ ] # Used Ratio: how much of credit limit is used by the customer i.e Utilization Ratio
    train_df['used_ratio'] = train_df['AVG_Bill_amt'] / train_df['LIMIT_BAL']
    valid_df['used_ratio'] = valid_df['AVG_Bill_amt'] / valid_df['LIMIT_BAL']
```

```
# average delay
pay_status_columns = [f'pay_{i}' for i in [0, 2, 3, 4, 5, 6]]
train_df['pay_mean_status'] = train_df[pay_status_columns].mean(axis=1)
valid_df['pay_mean_status'] = valid_df[pay_status_columns].mean(axis=1)

# Number of months with delayed payments
train_df['delayed_months'] = (train_df[pay_status_columns] >= 1).sum(axis=1)
valid_df['delayed_months'] = (valid_df[pay_status_columns] >= 1).sum(axis=1)
```

I created some new columns like **Used ratio** that gives me the idea about how much credit limit is used by customer also **average delay** that tells average delay over last 6 months **and also delayed months** that tells us no. of months in which payment was delayed

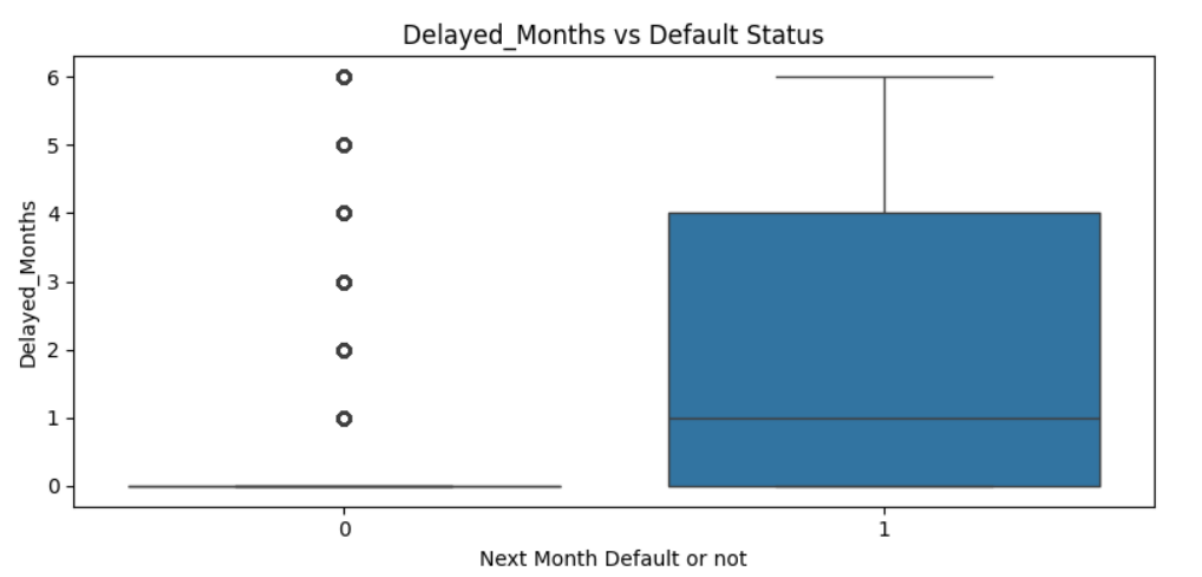
```
# Some of these columns are super skewed, so applying log helps smooth them out.
# Using np.log1p handles zero values safely, so we don't run into math issues.
# This makes the data easier for models like logistic regression or decision trees to work with.
Bill_cols = ['Bill_amt1','Bill_amt2', 'Bill_amt3', 'Bill_amt4', 'Bill_amt5', 'Bill_amt6']
for col in Bill_cols:
    train_df[f'log_{col}'] = np.log1p(train_df[col])
    valid_df[f'log_{col}'] = np.log1p(valid_df[col])

train_df[[f'log_{col}' for col in Bill_cols]].head()
```

	log_Bill_amt1	log_Bill_amt2	log_Bill_amt3	log_Bill_amt4	log_Bill_amt5	log_Bill_amt6
0	11.007500	11.014180	10.941531	10.261267	10.007511	9.940380
1	9.319972	0.000000	0.631272	0.500775	7.901733	7.208822
2	10.777538	10.798001	10.820483	10.834665	10.857143	10.877906
3	11.420300	11.439462	11.414798	11.353889	11.294990	11.250865
4	9.886718	6.907795	8.066986	10.712938	7.650059	0.854415

```
[ ] log_columns = [f'log_Bill_amt{i}' for i in range(1, 7)]
    train_df['avg_log_Bill_amt'] = train_df[log_columns].mean(axis=1)
    valid_df['avg_log_Bill_amt'] = valid_df[log_columns].mean(axis=1)
```

Also created some more columns like log_Bill_amt that smooths out the smooths out the super skewness of these columns and also avg_log_bill_amt

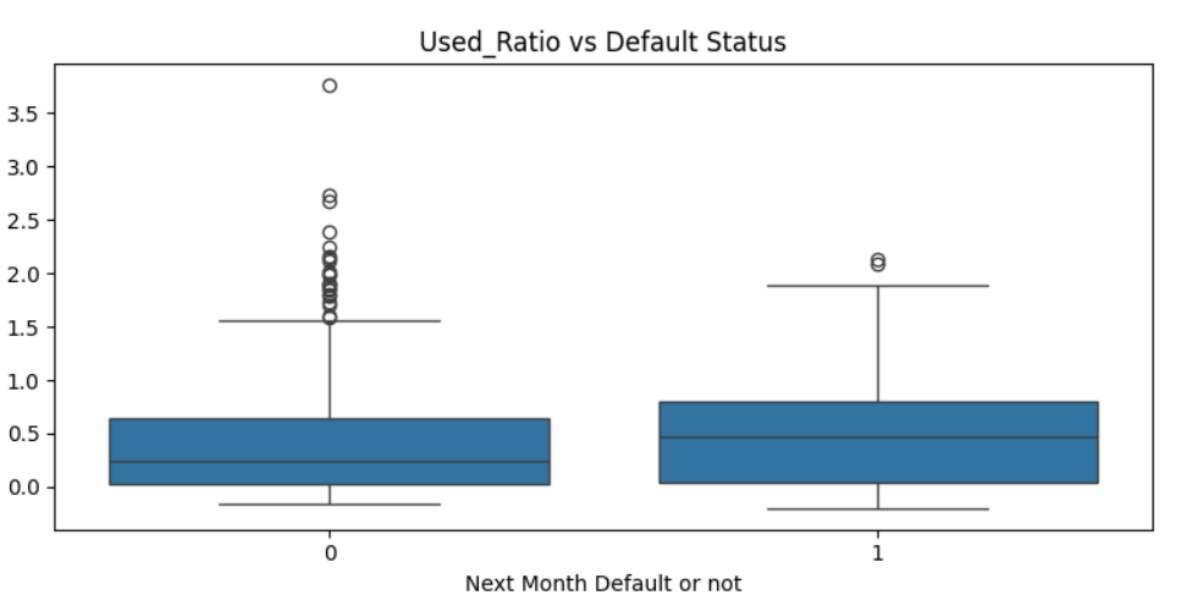
Now I plotted box plots of these new columns and observed a pattern in the boxplot of delayed months as you can see below:



So We can see guys who are going to default are delaying there payments

Also if we observe in the boxplot of Used_ratio vs default Status

then defaulters are using slightly more money than non defaulters



STEP-4

Data preparation and Balancing-

To ensure reliable model performance, we followed a structured approach for data cleaning, scaling, and balancing:

- **Feature Selection and Cleaning:**

Only numerical features were selected for modeling. Irrelevant columns such as `customer_ID` and the target variable were removed. Infinite values were replaced with NaNs and imputed using the median of each column

- **Scaling:**

Features were standardized using `StandardScaler` to ensure all variables contributed equally during model training. This step is particularly important for distance-based models and resampling methods.

- **Handling Class Imbalance:**

next_month_default	proportion
0	0.809601
1	0.190399

Since the dataset was imbalanced (with far fewer defaulters), we used a two-step resampling strategy:

  - **SMOTE (Synthetic Minority Oversampling Technique):** Applied to increase the number of defaulters to 40% of the total.
  - **SMOTE-Tomek Links:** Applied next to remove overlapping or borderline samples between classes, improving class separation.

- **Train-Test Split:**

The resampled dataset was split into training and validation sets using an 80-20 ratio to evaluate generalization performance**.**

- **Code For this is :**

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek
from collections import Counter
# DATA PREPARATION
X = train_df.drop(['Customer_ID', 'next_month_default'], axis=1)
X = X.select_dtypes(include=['number'])
X.replace([np.inf, -np.inf], np.nan, inplace=True)
X = X.fillna(X.median(numeric_only=True))
y = train_df['next_month_default'].loc[X.index]
# Scale features before resampling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Resampling
smote = SMOTE(sampling_strategy=0.4, random_state=101)
X_res, y_res = smote.fit_resample(X_scaled, y)

smt_f = SMOTETomek(random_state=42)
X_final, y_final = smt_f.fit_resample(X_res, y_res)

# Train-test split
X_train, X_val, y_train, y_val = train_test_split(X_final, y_final, test_size=0.2, random_state=101)
```

Also we can see the resampling effect and the shape of training and validating dataset after these operations

```
print("resampling Effect on class distribution", Counter(y_final))
print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
```

```
resampling Effect on class distribution Counter({0: 20163, 1: 20163})
Training set shape: (32260, 34)
Validation set shape: (8066, 34)
```

Step-5

. Model Training and Evaluation

To identify potential defaulters accurately, we trained and evaluated four classification models:

- **Logistic Regression**
- **Decision Tree Classifier**

- **Random Forest Classifier**
- **XGBoost Classifier**

These models were selected to cover both linear and non-linear decision boundaries, as well as tree-based ensembles that handle complex patterns and feature interactions.

EVALUATION METRICS:

Models were evaluated using:

- **F1 Score** – balances precision and recall
- **F2 Score** – emphasizes recall (important for catching defaulters)
- **ROC AUC Score** – measures overall class separation capability

The F2 score was prioritized because false negatives (missed defaulters) are more costly than false positives in the banking domain.

```
models = {
    'XGBoost': xgb_model,
    'Logistic Regression': logreg_model,
    'Decision Tree': dt_model,
    'Random Forest': rf_model
}

# Slightly changed thresholds for variety
thresholds = [0.55, 0.45, 0.38, 0.28]

for name, model in models.items():
    y_prob = model.predict_proba(X_val_scaled)[:, 1]
    print(f"\n{name} Model Threshold Tuning:")
    for thresh in thresholds:
        y_pred_thresh = (y_prob >= thresh).astype(int)
        print(f"\nThreshold: {thresh}")
        print(classification_report(y_val, y_pred_thresh, digits=4))
        print(f"ROC AUC Score: {roc_auc_score(y_val, y_prob):.4f}")
        print(f"F1 Score: {f1_score(y_val, y_pred_thresh):.4f}")
```

This part of code gives classification metric and f2 score and other parameters and then we observe that we can maximize the performance of our model by changing the threshold

So we define a function that gives us the best threshold for each model and also gives us the best model suitable

```python
def find_best_threshold(model, X_val_scaled, y_val, metric='f2'):
    y_prob = model.predict_proba(X_val_scaled)[:, 1]
    best_thresh = 0.5
    best_score = 0

    thresholds = np.arange(0.1, 0.9, 0.01)
    for thresh in thresholds:
        y_pred = (y_prob >= thresh).astype(int)
        if metric == 'f1':
            score = f1_score(y_val, y_pred)
        elif metric == 'f2':
            score = fbeta_score(y_val, y_pred, beta=2)
        else:
            raise ValueError("Metric must be 'f1' or 'f2'")

        if score > best_score:
            best_score = score
            best_thresh = thresh

    return best_thresh, best_score

# Find best thresholds for all models based on F2 score
best_thresholds = {}
for name, model in models.items():
    thresh, score = find_best_threshold(model, X_val_scaled, y_val, metric='f2')
    best_thresholds[name] = (thresh, score)
    print(f"{name} - Best Threshold: {thresh:.2f}, Best F2 Score: {score:.4f}")
```

```
XGBoost - Best Threshold: 0.64, Best F2 Score: 0.8341
Logistic Regression - Best Threshold: 0.22, Best F2 Score: 0.8328
Decision Tree - Best Threshold: 0.14, Best F2 Score: 0.8279
Random Forest - Best Threshold: 0.44, Best F2 Score: 0.8636
```

MODEL PERFORMANCE SUMMARY:

Model	F2 Score	ROC AUC	Notes
Logistic Regression	0.8328	0.77	Good baseline, limited in non-linearity
Decision Tree	0.8279	0.6923	Interpretable but prone to overfitting
Random Forest	0.8636	0.8476	Best balance of accuracy and robustness

Model	F2 Score	ROC AUC	Notes
XGBoost	0.8341	0.69	Strong recall, slightly lower AUC

So the best model is Random forest with the best F2 Score at a threshold of 0.44 so this will be our final

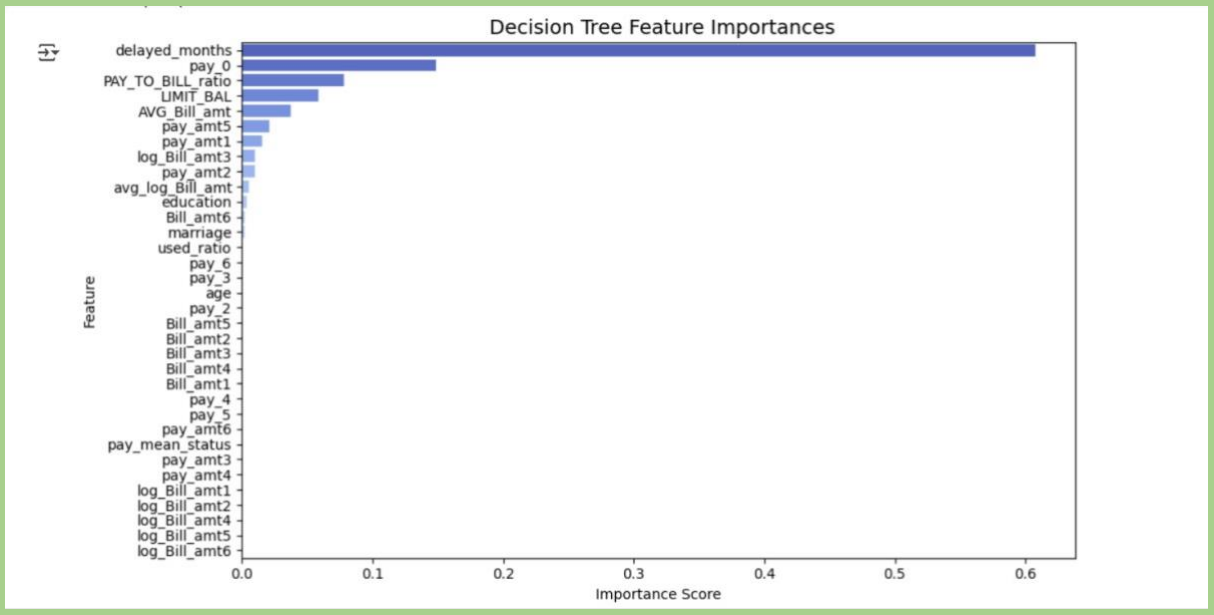
Feature Importance Analysis

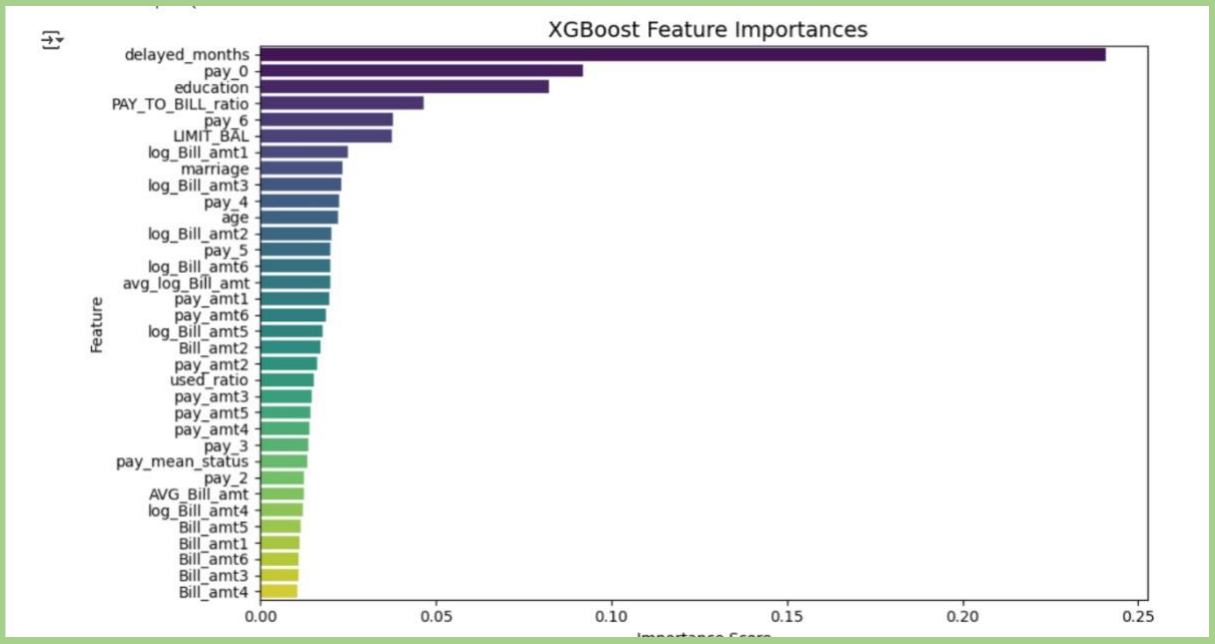
Feature importance was extracted from the tree-based models (Decision Tree, Random Forest, and XGBoost). Across these models, the following features consistently ranked as the most predictive:

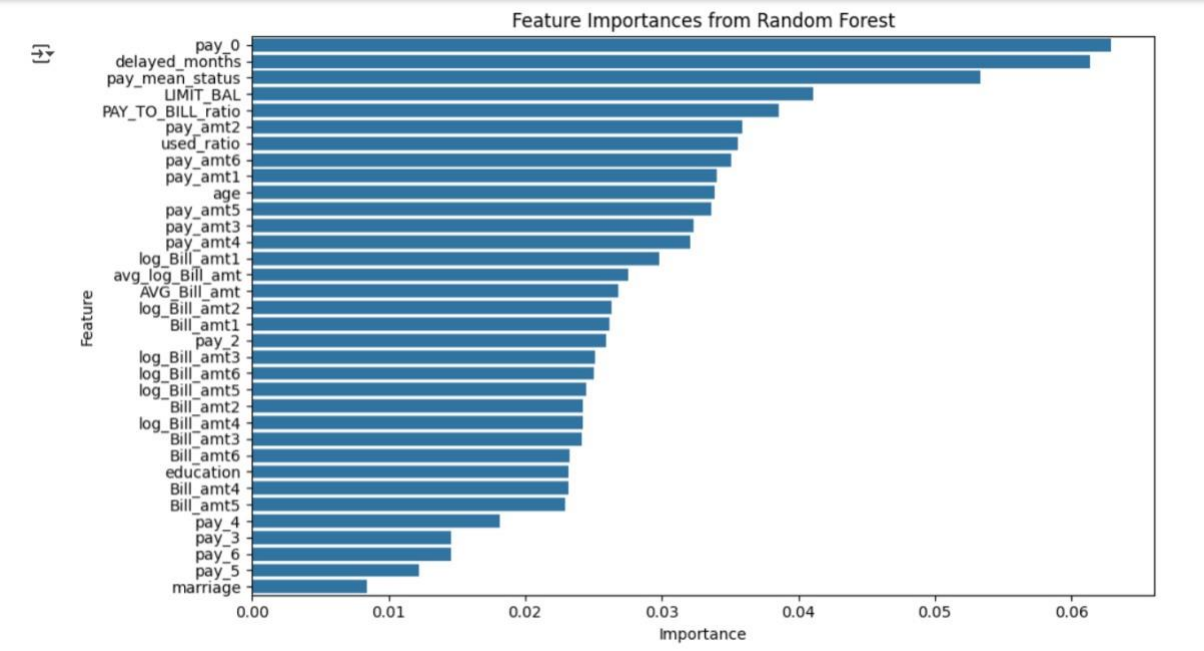
- `late_months` – Number of delayed payments
- `pay_0` – Most recent payment delay status
- `utilization_ratio` – Monthly credit usage relative to credit limit
- `limit_bal` – Total credit limit
- `avg_bill_amt` – Average bill over 6 months
- `repayment_ratio` – Payment as a fraction of bill amount

These features highlight that behavioral patterns, especially around payment history and credit utilization, are stronger indicators of default risk than static demographics like age or gender.

Also we see it from their graphs:-


Decision Tree Feature Importances
delayed_months
pay_0
PAY_TO_BILL_ratio
LIMIT_BAL
AVG_Bill_amt
pay_amt5
pay_amt1
log_Bill_amt3
pay_amt2
avg_log_Bill_amt
education
Bill_amt6
marriage
used_ratio
pay_6
pay_3
age
pay_2
Bill_amt5
Bill_amt2
Bill_amt3
Bill_amt4
Bill_amt1
pay_4
pay_5
pay_amt6
pay_mean_status
pay_amt3
pay_amt4
log_Bill_amt1
log_Bill_amt2
log_Bill_amt4
log_Bill_amt5
log_Bill_amt6
Feature
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Importance Score


XGBoost Feature Importances
delayed_months
pay_0
education
PAY_TO_BILL_ratio
pay_6
LIMIT_BAL
log_Bill_amt1
marriage
log_Bill_amt3
pay_4
age
log_Bill_amt2
pay_5
log_Bill_amt6
avg_log_Bill_amt
pay_amt1
pay_amt6
log_Bill_amt5
Bill_amt2
pay_amt2
used_ratio
pay_amt3
pay_amt5
pay_amt4
pay_3
pay_mean_status
pay_2
AVG_Bill_amt
log_Bill_amt4
Bill_amt5
Bill_amt1
Bill_amt6
Bill_amt3
Bill_amt4
Feature
0.00
0.05
0.10
0.15
0.20
0.25



Now the last step :

```
best_threshold = 0.44

# Define feature columns based on training data
feature_columns = X.columns.tolist()

# Prepare validation features
X_valid = valid_df[feature_columns].copy()

# Handle infinite values and missing data
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_valid.fillna(X_valid.median(numeric_only=True), inplace=True)

# Scale validation features using the scaler fitted on training data
X_valid_scaled = scaler.transform(X_valid)

# Predict probabilities with the chosen final model (Random Forest here)
y_prob_valid = rf_model.predict_proba(X_valid_scaled)[:, 1]

# Apply the optimal threshold
y_pred_valid = (y_prob_valid >= best_threshold).astype(int)

# Create submission DataFrame
submission = pd.DataFrame({
    'Customer_ID': valid_df['Customer_ID'],
    'next_month_default': y_pred_valid
})

submission_filename = "submission_23115049.csv"
submission.to_csv(submission_filename, index=False)
print(f"Submission file '{submission_filename}' created successfully.")
```

Simply Predicted for the Validation dataset and stored into a csv File

Summary:-

This project focused on building a machine learning model to predict credit card defaults in the upcoming billing cycle using customer behavioral and financial data. After thorough data cleaning, feature engineering, and

class balancing, four different models were trained: Logistic Regression, Decision Tree, Random Forest, and XGBoost.

Key insights from EDA highlighted that behavioral features such as payment delays, credit utilization, and repayment consistency were much more predictive of defaults than demographic variables. Among all models tested, the Random Forest classifier achieved the best performance based on F2 score and ROC AUC, making it the most suitable for early defaulter identification.

The final model provides a practical and interpretable tool for credit risk management. It allows financial institutions to take proactive measures such as limit adjustments and customer monitoring, ultimately reducing default-related losses and improving portfolio quality.

THANKING YOU