Presented By: Malay Kumar Parida

Pre-Approved Personal Loan Project

Data Analyst | Pre-Approved Personal Loan Lead Optimization in Banking Sector | SQL, Python, Predictive Modeling

https://www.linkedin.com/in/malay-kumar-parida/
malayparida96@gmail.com

- I built a full, end-to-end *“Banking – Rural Personal Loan Leads”* project that integrated **MySQL (DDL, EDA, joins)** with **Python (pandas) data handling**.
- The handling process imputed missing values using **mode for categorical** and **median for numeric features**, and capped outliers to ensure data quality.
- Using this cleaned dataset, I performed analysis to **generate a list of high-quality pre-approved leads** and **identify customers most likely to accept personal loan offers**, which helped improve conversion rates and significantly reduce non-responsive calling efforts.

I'll cover:

01 Database creation in MySQL

02 Table creation and data loading

03 Data cleaning using Python (mode/median fill)

04 Exploratory Data Analysis (EDA) using MySQL queries

05 Lead generation logic to identify potential pre-approved customers for personal loans

06 Identify customers most likely to accept personal loan offers

Data used

01 Demographics & occupation (customers)

02 Financials & DTI (financial_info, accounts)

03 CIBIL (cibil_scores)

04 Loan history/DPD/defaults (loans, external_loans)

05 Internal risk & PD (risk_scores)

06 6-month bank statements (bank_statements)

07 FD holdings (fixed_deposits)

07 Call response behavior (call_logs)

1. Create Database and Tables in MySQL

```sql
-- Step 1: Create a database
CREATE DATABASE axis_bank_loan;
USE axis_bank_loan;
```

```sql
-- Step 2: Create tables (simplified schema)
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    gender VARCHAR(10),
    occupation VARCHAR(100),
    rural_flag TINYINT(1)
);
```

```sql
CREATE TABLE risk_scores (
    customer_id INT,
    risk_score DECIMAL(5,2),
    delinquency_count INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```sql
CREATE TABLE loans (
    loan_id INT PRIMARY KEY,
    customer_id INT,
    loan_amount DECIMAL(15,2),
    loan_status VARCHAR(20), -- e.g., 'closed', 'defaulted', 'active'
    bank_source VARCHAR(50), -- 'axis' or 'external'
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```sql
CREATE TABLE call_logs (
    call_id INT PRIMARY KEY,
    customer_id INT,
    num_calls INT,
    last_call_date DATE,
    response_flag TINYINT(1),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```sql
CREATE TABLE financial_info (
    customer_id INT,
    annual_income DECIMAL(15,2),
    debt_to_income DECIMAL(5,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```sql
CREATE TABLE bank_statements (
    statement_id INT PRIMARY KEY,
    customer_id INT,
    month DATE,
    avg_balance DECIMAL(15,2),
    num_transactions INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```sql
CREATE TABLE cibil_scores (
    customer_id INT,
    cibil_score INT,
    last_updated DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```sql
CREATE TABLE fixed_deposits (
    fd_id INT PRIMARY KEY,
    customer_id INT,
    fd_amount DECIMAL(15,2),
    fd_maturity_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

2. Import Data

Load the CSV files into MySQL using:

```
LOAD DATA INFILE '/path/customers.csv'
INTO TABLE customers
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

(Repeat for all tables.)

3. Handle Missing Values Using Python (Mode & Median)

```
import pandas as pd

# Load data
customers = pd.read_csv('customers.csv')
financial_info = pd.read_csv('financial_info.csv')
cibil_scores = pd.read_csv('cibil_scores.csv')
risk_scores = pd.read_csv('risk_scores.csv')

# Fill categorical missing values with mode
customers['gender'].fillna(customers['gender'].mode()[0], inplace=True)
customers['occupation'].fillna(customers['occupation'].mode()[0], inplace=True)

# Fill numerical missing values with median
financial_info['annual_income'].fillna(financial_info['annual_income'].median(), inplace=True)
financial_info['monthly_income'].fillna(financial_info['monthly_income'].median(), inplace=True)
cibil_scores['cibil_score'].fillna(cibil_scores['cibil_score'].median(), inplace=True)
risk_scores['risk_score'].fillna(risk_scores['risk_score'].median(), inplace=True)

# Save cleaned data
customers.to_csv('customers_clean.csv', index=False)
financial_info.to_csv('financial_info_clean.csv', index=False)
cibil_scores.to_csv('cibil_scores_clean.csv', index=False)
risk_scores.to_csv('risk_scores_clean.csv', index=False)
```

4. Outlier Detection & Treatment (Using Boxplots)

```
import seaborn as sns
import matplotlib.pyplot as plt

# Boxplot for annual income
sns.boxplot(x=financial_info['annual_income'])
plt.show()

# Remove extreme outliers or cap them
q1 = financial_info['annual_income'].quantile(0.25)
q3 = financial_info['annual_income'].quantile(0.75)
iqr = q3 - q1
upper_limit = q3 + 1.5 * iqr
financial_info['annual_income'] = financial_info['annual_income'].apply(lambda x: upper_limit if x > upper_limit else x)
```

5. Exploratory Data Analysis (EDA) in MySQL

Check data size

```
SELECT COUNT(*) AS total_customers FROM customers;
SELECT COUNT(*) AS total_loans FROM loans;
```

Check for Missing Values

```
SELECT COUNT(*) FROM customers WHERE gender IS NULL;
SELECT COUNT(*) FROM financial_info WHERE annual_income IS NULL;
```

Distribution of CIBIL Scores

```
SELECT cibil_score, COUNT(*) AS cnt
FROM cibil_scores
GROUP BY cibil_score
ORDER BY cibil_score;
```

Average Income by Occupation

```
SELECT occupation, ROUND(AVG(annual_income),2) AS avg_income
FROM financial_info
JOIN customers USING (customer_id)
GROUP BY occupation;
```

Basic demographics

```
SELECT gender, COUNT(*) AS cnt
FROM customers
GROUP BY gender;

SELECT occupation, COUNT(*) AS cnt
FROM customers
GROUP BY occupation
ORDER BY cnt DESC;
```

Financial health

```
SELECT
    ROUND(AVG(annual_income),2) AS avg_income,
    ROUND(AVG(debt_to_income),2) AS avg_dti
FROM financial_info;
```

Loan default patterns

```
SELECT loan_status, COUNT(*) AS cnt
FROM loans
GROUP BY loan_status;
```

Identify high-risk customers (low CIBIL + high delinquency)

```
SELECT c.customer_id, c.cibil_score, r.delinquency_count
FROM cibil_scores c
JOIN risk_scores r ON c.customer_id = r.customer_id
WHERE c.cibil_score < 650 AND r.delinquency_count > 2;
```

6. Identify Customers for Pre-Approved Loans

Business logic:
- Age between **25 and 55**
- CIBIL score ≥ 750
- Debt-to-Income Ratio < 40%
- No default history
- Risk score ≤ 3.0

```
SELECT c.customer_id, c.name, f.annual_income, cs.cibil_score, r.risk_score
FROM customers c
JOIN financial_info f ON c.customer_id = f.customer_id
JOIN cibil_scores cs ON c.customer_id = cs.customer_id
JOIN risk_scores r ON c.customer_id = r.customer_id
WHERE c.age BETWEEN 25 AND 55
  AND cs.cibil_score >= 750
  AND f.debt_to_income_ratio < 40
  AND r.delinquency_count = 0
  AND r.risk_score <= 3.0;
```

7. Lead Generation Final Output

we can even **create a view** for pre-approved leads:

This is Final Output =>

```
CREATE VIEW pre_approved_leads AS
SELECT c.customer_id, c.name, f.annual_income, cs.cibil_score, r.risk_score
FROM customers c
JOIN financial_info f ON c.customer_id = f.customer_id
JOIN cibil_scores cs ON c.customer_id = cs.customer_id
JOIN risk_scores r ON c.customer_id = r.customer_id
WHERE c.age BETWEEN 25 AND 55
  AND cs.cibil_score >= 750
  AND f.debt_to_income_ratio < 40
  AND r.delinquency_count = 0
  AND r.risk_score <= 3.0;
```

Second Approach

Step 1: Select rural customers with good credit

```
SELECT cu.customer_id, cu.age, fi.annual_income, c.cibil_score, r.risk_score
FROM customers cu
JOIN financial_info fi ON cu.customer_id = fi.customer_id
JOIN cibil_scores c ON cu.customer_id = c.customer_id
JOIN risk_scores r ON cu.customer_id = r.customer_id
WHERE cu.rural_flag = 1
  AND c.cibil_score >= 700
  AND r.risk_score >= 70;
```

Step 2: Exclude customers with active loans

```
SELECT cu.customer_id, cu.age, fi.annual_income, c.cibil_score, r.risk_score
FROM customers cu
JOIN financial_info fi ON cu.customer_id = fi.customer_id
JOIN cibil_scores c ON cu.customer_id = c.customer_id
JOIN risk_scores r ON cu.customer_id = r.customer_id
LEFT JOIN loans l ON cu.customer_id = l.customer_id AND l.loan_status = 'active'
WHERE cu.rural_flag = 1
  AND c.cibil_score >= 700
  AND r.risk_score >= 70
  AND l.loan_id IS NULL;
```

Step 3: Exclude non-responsive customers

```
SELECT final.customer_id, final.age, final.annual_income, final.cibil_score, final.risk_score
FROM (
    SELECT cu.customer_id, cu.age, fi.annual_income, c.cibil_score, r.risk_score
    FROM customers cu
    JOIN financial_info fi ON cu.customer_id = fi.customer_id
    JOIN cibil_scores c ON cu.customer_id = c.customer_id
    JOIN risk_scores r ON cu.customer_id = r.customer_id
    LEFT JOIN loans l ON cu.customer_id = l.customer_id AND l.loan_status = 'active'
    WHERE cu.rural_flag = 1
      AND c.cibil_score >= 700
      AND r.risk_score >= 70
      AND l.loan_id IS NULL
) AS final
LEFT JOIN call_logs cl ON final.customer_id = cl.customer_id
WHERE cl.num_calls < 7 OR cl.response_flag = 1;
```

Lead generation logic:

- Rural customers
- Good CIBIL (>700) & high risk score (>70)
- No active loans
- Responsive to calls
- Final list of high-quality pre-approved leads helps reduce unnecessary calling efforts.

Some Extra EDA Questions

Non-responders (>7 consecutive "No Answer/Busy"):

```sql
SELECT customer_id, MAX(nr_streak) AS max_nr_streak, COUNT(*) AS total_calls
FROM call_logs
GROUP BY customer_id
HAVING MAX(nr_streak) > 7
ORDER BY max_nr_streak DESC;
```

Six-month inflow/outflow per customer:

```sql
SELECT customer_id,
       SUM(CASE WHEN txn_type='CREDIT' THEN amount ELSE 0 END) AS inflow_6m,
       SUM(CASE WHEN txn_type='DEBIT' THEN amount ELSE 0 END) AS outflow_6m
FROM bank_statements
GROUP BY customer_id;
```

Risk vs default:

```sql
SELECT r.risk_bucket,
       ROUND(AVG(l.dpd_max),2) AS avg_dpd_max,
       ROUND(AVG(l.default_flag),3) AS default_rate
FROM risk_scores r
JOIN loans l USING(customer_id)
GROUP BY r.risk_bucket;
```

Conclusion & Key Takeaways

- The project successfully analyzed rural personal loan data of Banking customers to identify potential leads for pre-approved loans.
- Using MySQL for data integration and Python (Pandas) for cleaning and preprocessing, we handled missing values (mode for categorical, median for numerical) and removed outliers using boxplot analysis.
- After processing 500+ records in each table, we joined datasets (demographics, financial info, CIBIL, risk score, FD data, call logs, bank statements) to create a unified customer view.
- The final dataset helped reduce unnecessary calling efforts by identifying high-potential customers, thus improving lead quality and saving operational costs.

Business Impact

- Improved targeting: Focused on customers most likely to take a personal loan.
- Cost savings: Reduced wasted calls to non-responsive customers (>7 calls).
- Better decision-making: Provided a structured SQL pipeline for analytics teams.

Thank You

Presented By:
Malay Kumar Parida

https://www.linkedin.com/in/malay-kumar-parida/
malayparida96@gmail.com