Credit Risk Modelling Project Report

Problem Statement: The objective of this project is to build a predictive model that can accurately determine the likelihood of a customer defaulting on a loan. This model aims to assist in making well-informed loan approval decisions, ultimately reducing financial risk for the lending institution.

About the Dataset

We have two datasets:

1. **Product Dataset** (`df1`)
2. **CIBIL Dataset** (`df2`)

```
[8] df1.head()
```

	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	Total_TL_opened_L6M	Tot_TL_closed_L6M	pct_tl_open_L6M	pct_tl_closed_L6M	pct_active_tl
0	1	5	4	1	0	0	0.000	0.0	0.200
1	2	1	0	1	0	0	0.000	0.0	1.000
2	3	8	0	8	1	0	0.125	0.0	1.000
3	4	1	0	1	1	0	1.000	0.0	1.000
4	5	3	2	1	0	0	0.000	0.0	0.333

5 rows × 26 columns

```
df1.shape
```

```
(51336, 26)
```

```
[11] df2.head()
```

	PROSPECTID	time_since_recent_payment	time_since_first_deliquency	time_since_recent_deliquency	num_times_delinquent	max_delinquency_level
0	1	549	35	15	11	29
1	2	47	-99999	-99999	0	-99999
2	3	302	11	3	9	25
3	4	-99999	-99999	-99999	0	-99999
4	5	583	-99999	-99999	0	-99999

5 rows × 62 columns

```
df2.shape
```

```
(51336, 62)
```

Handling Null Values

1. **Identification and Removal of Columns with Excessive Missing Values**

- Many columns in the datasets have missing values represented as `-99999`. To address this, we first identified columns with more than 10,000 missing values.
- Columns with such a high number of missing values were dropped from the dataset, as they are unlikely to provide useful information.
-

2. **Handling Rows with Missing Values**
   - After removing columns with excessive missing values, we proceeded to handle rows with remaining null values.
   - Rows containing any missing values were dropped from the dataset to ensure the quality and completeness of the data used for modeling.

-

Merging the Datasets

1. **Finding a Common Column**
   - To combine `df1` and `df2`, we identified a common column that exists in both datasets.
   - This common column served as the basis for merging the two dataframes.
2. **Merging the Dataframes**
   - We used the `pd.merge()` function from the pandas library to join `df1` and `df2` on the identified common column.
   - The resulting dataframe, which contains combined information from both datasets, is named `df`.

```
[23] df.shape
```

```
(42064, 79)
```

```
df.head()
```

	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	Total_TL_opened_L6M	Tot_TL_closed_L6M	pct_tl_open_L6M	pct_tl_closed_L6M	pct_active_tl
0	1	5	4	1	0	0	0.000	0.0	0.200
1	2	1	0	1	0	0	0.000	0.0	1.000
2	3	8	0	8	1	0	0.125	0.0	1.000
3	5	3	2	1	0	0	0.000	0.0	0.333
4	6	6	5	1	0	0	0.000	0.0	0.167

5 rows × 79 columns

Feature Analysis and Selection

1. **Categorical Feature Analysis**

- **Chi-Square Test**: We applied the Chi-Square test to evaluate the association between categorical features and the target variable (loan default status). This test helps determine whether the observed frequencies of categorical data differ significantly from expected frequencies under the null hypothesis of no association. Features that were not significantly associated with the target variable were removed.

Test Statistics

H0: Ctegorical columns are not associated with Approved_Flag

H1: Ctegorical columns are associated with Approved_Flag

```
[27] #Chi_square_test
     for i in ['MARITALSTATUS','EDUCATION', 'GENDER', 'last_prod_enq2', 'first_prod_enq2']:
       chi2, pval, dof, expected = chi2_contingency(pd.crosstab(df[i],df['Approved_Flag']))
       print(i,'----', pval)
```

```
MARITALSTATUS ---- 3.578180861038862e-233
EDUCATION ---- 2.6942265249737532e-30
GENDER ---- 1.907936100186563e-05
last_prod_enq2 ---- 0.0
first_prod_enq2 ---- 7.84997610555419e-287
```

2. **Numerical Feature Analysis**
   - **Variance Inflation Factor (VIF)**: To assess multicollinearity among numerical features, we used the VIF. High multicollinearity can inflate the variance of coefficient estimates and lead to less reliable results. Features with high VIF values were removed to reduce redundancy.

```
# VIF sequentially check

vif_data = df[numeric_columns]
total_columns = vif_data.shape[1]
columns_to_be_kept = []
column_index = 0



for i in range (0,total_columns):

    vif_value = variance_inflation_factor(vif_data, column_index)
    print (column_index,'---',vif_value)


    if vif_value <= 6:
        columns_to_be_kept.append( numeric_columns[i] )
        column_index = column_index+1

    else:
        vif_data = vif_data.drop([ numeric_columns[i] ] , axis=1)
```

- **ANOVA Test**: We performed the ANOVA test to analyze the differences between numerical feature groups with respect to the target variable. ANOVA helps in identifying whether there are statistically significant differences in means among different groups. Numerical features that did not show significant differences were dropped.

```
# Check Anova for columns_to_be_kept

from scipy.stats import f_oneway

columns_to_be_kept_numerical=[]

for i in columns_to_be_kept:
  a= list(df[i])
  b= list(df['Approved_Flag'])

  group_P1=[value for value, group in zip(a,b) if group =='P1']
  group_P2=[value for value, group in zip(a,b) if group =='P2']
  group_P3=[value for value, group in zip(a,b) if group =='P3']
  group_P4=[value for value, group in zip(a,b) if group =='P4']

  f_statistics, p_value =f_oneway(group_P1, group_P2, group_P3,group_P4)

  if p_value<0.05:
    columns_to_be_kept_numerical.append(i)
    print(i)
```

3. **Label Encoding**
   - Categorical variables were converted into a numerical format using label encoding. This step is necessary for machine learning algorithms to process categorical data effectively.

Machine Learning Model Fitting

1. **Data Splitting**
   - The dataset was split into dependent (target) and independent (features) variables.
   - We then divided the data into training and test sets to evaluate the performance of our models.
2. **Model Training**
   - We trained several machine learning models to predict loan default:
     - **Random Forest**
     - **XGBoost**
     - **Decision Tree**
   - Among these models, XGBoost achieved the highest accuracy of 0.79.

Conclusion

The final model, XGBoost, demonstrated the best performance with an accuracy of 0.79. This model will help in predicting the likelihood of a customer defaulting on a loan, thereby supporting better decision-making and reducing financial risk for the lending institution.