



# Enhancing Credit Risk Assessment through Advanced Machine Learning Techniques

Phase 2 Review 2



# Project Group No : 244

Group Supervisor – Dr . Preetam Suman

## Group Members

- ❑ 21BCE10443 Harshit Varshney
- ❑ 21BCE11188 Adarsh Kanungo
- ❑ 21BCE10470 Giya Ambasta
- ❑ 21BCE10503 Saumadipta Chatterjee

# Objective

**Aim:** The objective is to predict whether to approve or deny a loan application using customer data.

**Purpose:**

- Optimize the bank's lending strategy.
- Reduce the occurrence of non-performing loans.
- Assist the bank in making data-driven credit decisions.

**Motivation:**

- Address existing gaps in credit risk modeling.

# Clarity and Significance of the Problem Statement

- **Problem Statement:**
  - The bank faces challenges in accurately predicting credit risk for new and existing customers, leading to increased bad loans and impacting profitability.
- **Significance:**
  - Addressing this issue with advanced modeling will reduce loan defaults, improve profitability, and enhance customer profiling.

# Feasibility and Scope of the Project

- **Feasibility:**
  - Leverages available internal and external datasets (Bureau dataset, Internal product dataset).
  - Machine learning models can be deployed to automate risk assessment.
- **Scope:**
  - Improves risk assessment for credit cards, personal loans, housing loans, etc.
  - Can be scaled to other products and sectors of the bank's business.

# Existing Projects

## Similar Projects:

- Previous credit scoring systems rely on rule-based models or traditional scoring techniques like logistic regression.

## Difference:

- Used 2 datasets 1.Bank Internal dataset 2.Cibil dataset for more accurate results
- Proposed model uses advanced ML techniques (e.g., random forest, XGBoost) to enhance prediction accuracy.

## Initial Research Findings:

- Higher accuracy in predicting delinquencies compared to existing models.
- Improved risk segmentation based on customer behavior and profile data.

# Some Basic Banking Terms

**Asset:** Loans like housing, personal, vehicle, and education loans. These are the bank's assets because they generate income through interest payments.

**Liability:** Products like current accounts, savings accounts, fixed deposits, and RDs. These are considered liabilities because the bank must pay interest on these accounts.

**NPA (Non-Performing Asset):** A loan is considered NPA when payments are overdue (Days Past Due, DPD) for more than 90 days. It indicates that the loan is in default.

**Disbursed Amount:** The amount of the loan granted to the customer.

**Outstanding Principal (OSP):** The remaining loan balance that the customer still owes. Ideally, it should be zero at the end of the loan term.

**DPD (Days Past Due):** The number of days a payment has been delayed. DPD should ideally be zero, meaning no payment delay.

**PAR (Portfolio at Risk):** The outstanding principal at risk when DPD is greater than zero, indicating potential for default.

### Credit Risk Types:

- **Non-delegate account (NDA):** DPD = 0, meaning no payment delays.
- **SMA1 (Special Mention Account 1):** DPD is between 0-30 days.
- **SMA2 (Special Mention Account 2):** DPD is between 31-60 days.
- **SMA3 (Special Mention Account 3):** DPD is between 61-90 days.
- **NPA (Non-Performing Asset):** DPD is between 90-180 days.
- **Written Off:** DPD is greater than 180 days, meaning the loan is deleted from the bank's books, but recovery may still be pursued.

### NPA Categories:

- **GNPA (Gross NPA):** The total amount of NPA, typically between 3-5% of the bank's outstanding principal.
- **NNPA (Net NPA):** This is GNPA minus the provisioning amount (the amount set aside to cover potential losses), generally ranging from 0.01-0.06%. Lower NNPA reflects better financial health of the bank.



# Datasets & Their Feature Description

**Case Study 1:** Internal product dataset containing 25 features related to the bank's customer loan product holdings.

**Case Study 2:** CIBIL dataset with features to predict loan priority using CIBIL scoring rules (P1, P2, P3, P4).

# Datasets & Their Feature Description

**Case Study 1:** Internal product dataset containing 25 features related to the bank's customer loan product holdings.

Variable Name	Description
Total_TL	Total trade lines/accounts in Bureau
Tot_Closed_TL	Total closed trade lines/accounts
Tot_Active_TL	Total active accounts
Total_TL_opened_L6M	Total accounts opened in last 6 Months
Tot_TL_closed_L6M	Total accounts closes in last 6 months
pct_tl_open_L6M	Percent accounts opened in last 6 months
pct_tl_closed_L6M	percent accounts closed in last 6 months
pct_active_tl	Percent active accounts
pct_closed_tl	Percent closed accounts
Total_TL_opened_L12M	Total accounts opened in last 12 Months
Tot_TL_closed_L12M	Total accounts closed in last 12 months

pct_tl_open_L12M	Percent accounts opened in last 12 months
pct_tl_closed_L12M	percent accounts closed in last 12 months
Tot_Missed_Pmnt	Total missed Payments
Auto_TL	Count Automobile accounts
CC_TL	Count of Credit card accounts
Consumer_TL	Count of Consumer goods accounts
Gold_TL	Count of Gold loan accounts
Home_TL	Count of Housing loan accounts
PL_TL	Count of Personal loan accounts
Secured_TL	Count of secured accounts
Unsecured_TL	Count of unsecured accounts
Other_TL	Count of other accounts
Age_Oldest_TL	Age of oldest opened account
Age_Newest_TL	Age of newest opened account

## Case Study 2: CIBIL dataset with features to predict loan priority using CIBIL scoring rules (P1, P2, P3, P4).

time_since_recent_payment	Time Since recent Payment made
time_since_first_delinquency	Time since first Delinquency (missed payment)
time_since_recent_delinquency	Time Since recent Delinquency
num_times_delinquent	Number of times delinquent
max_delinquency_level	Maximum delinquency level
max_recent_level_of_deliq	Maximum recent level of delinquency
num_deliq_6mts	Number of times delinquent in last 6 months
num_deliq_12mts	Number of times delinquent in last 12 months
num_deliq_6_12mts	Number of times delinquent between last 6 months and last 12 months
max_deliq_6mts	Maximum delinquency level in last 6 months
max_deliq_12mts	Maximum delinquency level in last 12 months
num_times_30p_dpd	Number of times 30+ dpd
num_times_60p_dpd	Number of times 60+ dpd
num_std	Number of standard Payments
num_std_6mts	Number of standard Payments in last 6 months
num_std_12mts	Number of standard Payments in last 12 months
num_sub	Number of sub standard payments - not making full payments
num_sub_6mts	Number of sub standard payments in last 6 months
num_sub_12mts	Number of sub standard payments in last 12 months
num_dbt	Number of doubtful payments
num_dbt_6mts	Number of doubtful payments in last 6 months
num_dbt_12mts	Number of doubtful payments in last 12 months

num_lss	Number of loss accounts
num_lss_6mts	Number of loss accounts in last 6 months
num_lss_12mts	Number of loss accounts in last 12 months
recent_level_of_delinq	Recent level of delinquency
tot_enq	Total enquiries
CC_enq	Credit card enquiries
CC_enq_L6m	Credit card enquiries in last 6 months
CC_enq_L12m	Credit card enquiries in last 12 months
PL_enq	Personal Loan enquiries
PL_enq_L6m	Personal Loan enquiries in last 6 months
PL_enq_L12m	Personal Loan enquiries in last 12 months
time_since_recent_enq	Time since recent enquiry
enq_L12m	Enquiries in last 12 months
enq_L6m	Enquiries in last 6 months
enq_L3m	Enquiries in last 3 months
MARITALSTATUS	Marital Status
EDUCATION	Education level
AGE	Age
GENDER	
NETMONTHLYINCOME	
Time_With_Curr_Empr	Time with current Employer
pct_of_active_TLs_ever	Percent active accounts ever
pct_opened_TLs_L6m_of_L12m	Percent accounts opened in last 6 months to last 12 months
pct_currentBal_all_TL	Percent current balance of all accounts
CC_utilization	Credit card utilization
CC_Flag	Credit card Flag
PL_utilization	Personal Loan utilization
PL_Flag	Personal Loan Flag
pct_PL_enq_L6m_of_L12m	Percent enquiries PL in last 6 months to last 12 months
pct_CC_enq_L6m_of_L12m	Percent enquiries CC in last 6 months to last 12 months

pct_PL_enq_L6m_of_ever	Percent enquiries PL in last 6 months to last 6 months
pct_CC_enq_L6m_of_ever	Percent enquiries CC in last 6 months to last 6 months
max_unsec_exposure_inPct	Maximum unsecured exposure in percent
HL_Flag	Housing Loan Flag
GL_Flag	Gold Loan Flag
last_prod_enq2	Lates product enquired for
first_prod_enq2	First productd enquired for
Credit_Score	Applicant's credit score
Approved_Flag	Priority levels

# Feature Engineering & EDA

The Phase 1 of capstone project so far has focused on the initial and crucial steps of data processing, which include dataset cleaning, feature engineering, and exploratory data analysis (EDA). The main aim was to prepare the dataset for further analysis by handling missing values, identifying important features, and gaining insights into the data through visualizations and statistical methods.

The code primarily involves:

- ❖ **Data Cleaning:** Null values, represented as -99999, were identified and appropriately handled based on the banking requirements.
- ❖ **Feature Engineering:** Multiple features were transformed and prepared for analysis, ensuring the integrity and relevance of data.
- ❖ **Exploratory Data Analysis:** Insights were drawn from both categorical and numerical features, testing associations with the target variable using chi-square tests, and multicollinearity was checked with VIF analysis.

- The first step is to load the necessary Python libraries that help with data handling, statistical tests, and multicollinearity checks.

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named 'Feature engineering.py' with the following code:

```
1 # Import necessary libraries
2
3 import pandas as pd
4 from scipy.stats import chi2_contingency
5 from statsmodels.stats.outliers_influence import variance_inflation_factor
6
7
8
9
10
11
12 # Load the dataset
13 a1 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\a
14 a2 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\a
15
16
17
18
19 df1 = a1.copy()
20 df2 = a2.copy()
21
22
23
24
25
26
27 # Remove nulls
28 df1 = df1.loc[df1['Age_Oldest_TL'] != -99999]
```

The right-hand side of the IDE contains two panels. The top panel is the 'Variable Explorer', which is currently empty. The bottom panel is the 'Console', showing the execution of the code cells:

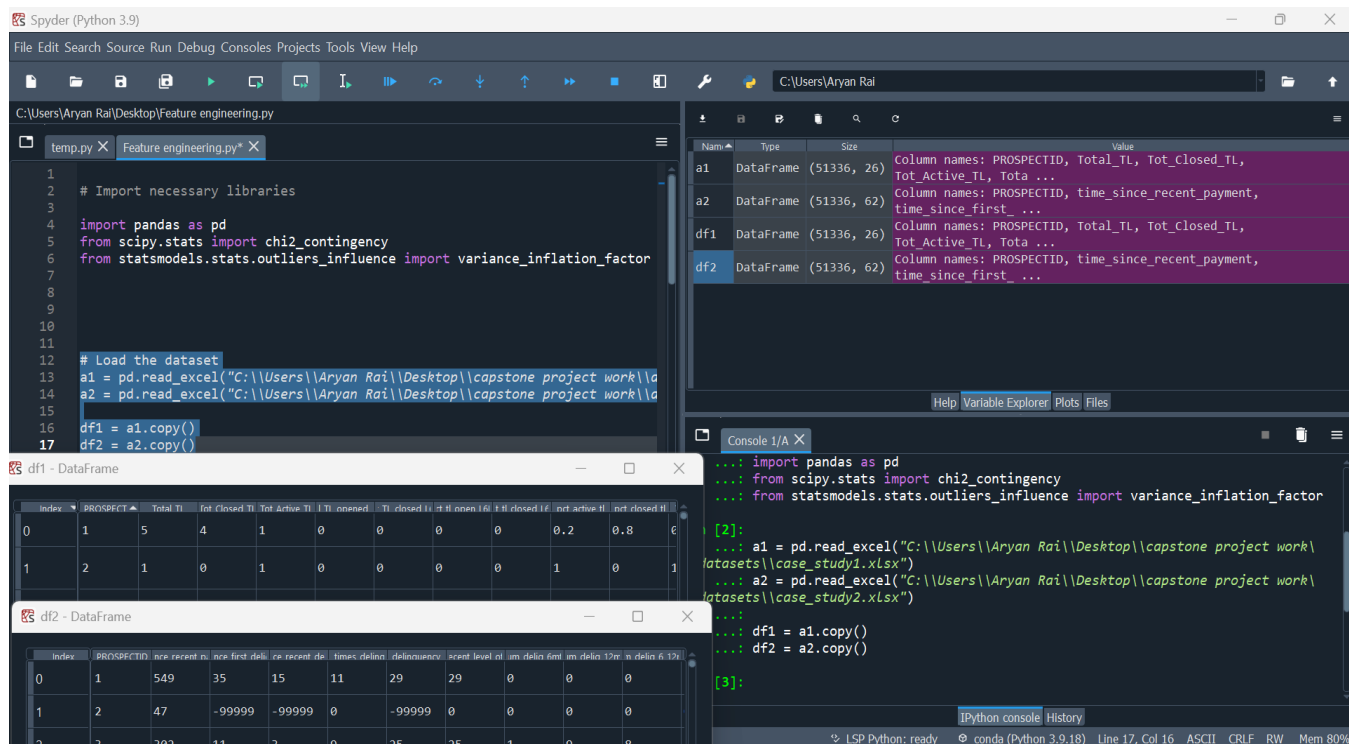
```
In [1]:
...: import pandas as pd
...: from scipy.stats import chi2_contingency
...: from statsmodels.stats.outliers_influence import variance_inflation_factor

In [2]:
```

The status bar at the bottom indicates the current environment is 'conda (Python 3.9.18)' and shows the file path 'C:\\Users\\Aryan Rai\\Desktop\\Feature engineering.py'.



- Two Excel files, case\_study1.xlsx (internal product data) and case\_study2.xlsx (CIBIL data), are loaded into Python and copied into dataframes df1 and df2 for further analysis.



```
1 # Import necessary libraries
2
3
4 import pandas as pd
5 from scipy.stats import chi2_contingency
6 from statsmodels.stats.outliers_influence import variance_inflation_factor
7
8
9
10
11
12 # Load the dataset
13 a1 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\d
14 a2 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\d
15
16 df1 = a1.copy()
17 df2 = a2.copy()
```

Variable Explorer

Name	Type	Size	Value
a1	DataFrame	(51336, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot ...
a2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_payment, time_since_first_ ...
df1	DataFrame	(51336, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot ...
df2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_payment, time_since_first_ ...

Console 1/A X

```
[2]: import pandas as pd
...: from scipy.stats import chi2_contingency
...: from statsmodels.stats.outliers_influence import variance_inflation_factor
[2]:
...: a1 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\
...: datasets\\case_study1.xlsx")
...: a2 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\
...: datasets\\case_study2.xlsx")
...: df1 = a1.copy()
...: df2 = a2.copy()
[3]:
```

df1 - DataFrame

index	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	TL_opened	TL_closed	TL_opened	TL_closed	TL_active	Tot_Closed_TL
0	1	5	4	1	0	0	0	0	0.2	0.8
1	2	1	0	1	0	0	0	0	1	0

df2 - DataFrame

index	PROSPECTID	nce_recent_p	nce_first_del	nce_recent_de	times_delinq	delinquency	acount_level	delinq_12m	delinq_12m	delinq_12m
0	1	549	35	15	11	29	29	0	0	0
1	2	47	-99999	-99999	0	-99999	0	0	0	0
2	3	302	11	3	0	25	25	1	0	0

- The code identifies missing values (represented as -99999) in both datasets. In df1, rows where the oldest account age is -99999 are removed. In df2, columns with more than 10,000 missing values are dropped, and remaining rows with null values are also removed.

The screenshot shows the Spyder Python IDE interface. The left pane displays a script named 'Feature engineering.py' with the following code:

```
21
22
23
24 # Remove nulls
25 df1 = df1.loc[df1['Age_Oldest_TL'] != -99999]
26
27
28
29
30 columns_to_be_removed = []
31
32 for i in df2.columns:
33     if df2.loc[df2[i] == -99999].shape[0] > 10000:
34         columns_to_be_removed.append(i)
35
36
37
38 df2 = df2.drop(columns_to_be_removed, axis=1)
39
40
41
42 for i in df2.columns:
43     df2 = df2.loc[df2[i] != -99999]
44
45
46
47
48
49
50 # Checking common column names
51 for i in list(df1.columns):
52     if i in list(df2.columns):
```

The right pane shows the Variable Explorer with the following data:

Name	Type	Size	Value
a1	DataFrame	(51336, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot ...
a2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_payment, time_since_first ...
df1	DataFrame	(51296, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot ...
df2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_payment, time_since_first ...

The bottom pane shows the IPython console with the following execution history:

```
In [2]:
...: a1 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\
...: datasets\\case_study1.xlsx")
...: a2 = pd.read_excel("C:\\Users\\Aryan Rai\\Desktop\\capstone project work\\
...: datasets\\case_study2.xlsx")
...:
...: df1 = a1.copy()
...: df2 = a2.copy()

In [3]:
...: df1 = df1.loc[df1['Age_Oldest_TL'] != -99999]

In [4]:
```

The status bar at the bottom indicates: LSP Python: ready, conda (Python 3.9.18), Line 25, Col 46, ASCII, CRLF, RW, Mem 81%.

- The two datasets are merged based on a common column PROSPECTID. This ensures that we have a unified dataset with no missing values, ready for analysis.

- The code then checks which columns contain categorical data (e.g., "Marital Status"). This is important for statistical tests later.

The screenshot shows the Spyder Python IDE interface. The main editor displays a Python script named 'feature\_engineering.py' with the following code:

```
56
57
58 # Merge the two dataframes, inner join so that no nulls are present
59 df = pd.merge ( df1, df2, how = 'inner', left_on = ['PROSPECTID'], right_on
60
61
62
63
64
65 # check how many columns are categorical
66 for i in df.columns:
67     if df[i].dtype == 'object':
68         print(i)
69
70
71
72
73
74
75
76 # Chi-square test
77 for i in ['MARITALSTATUS', 'EDUCATION', 'GENDER', 'last_prod_enq2', 'first_
78     chi2, pval, _, _ = chi2_contingency(pd.crosstab(df[i], df['Approved_Fla
79     print(i, '---', pval)
80
81
82
83
84
85
86
87
88
89 # VTE for numerical columns
```

The right-hand pane shows the 'Variable Explorer' with a table of variables:

Name	Type	Size	Value
a1	DataFrame	(51336, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
a2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_payment, time_since_first...
columns_to_be_removed	list	8	['time_since_first_delinquency', 'time_since_recent_delinquency', 'max_d...
df	DataFrame	(42064, 79)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
df1	DataFrame	(51296, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
df2	DataFrame	(42066, 54)	Column names: PROSPECTID, time_since_recent_payment, num_times_delinqu...
i	str	13	Approved_Flag

The bottom pane shows the 'Console' with the following output:

```
In [7]:
...: for i in df.columns:
...:     if df[i].dtype == 'object':
...:         print(i)
MARITALSTATUS
EDUCATION
GENDER
last_prod_enq2
first_prod_enq2
Approved_Flag

In [8]:
```

The status bar at the bottom indicates: LSP Python: ready, conda (Python 3.9.18), Line 68, Col 17, ASCII, CRLF, RW, Mem 80%.

# Multicollinearity

- Happens when two or more Independent Variables are **correlated**
- This leads to lower the accuracy of our model
- Needs to be removed by some method (eg- VIF)

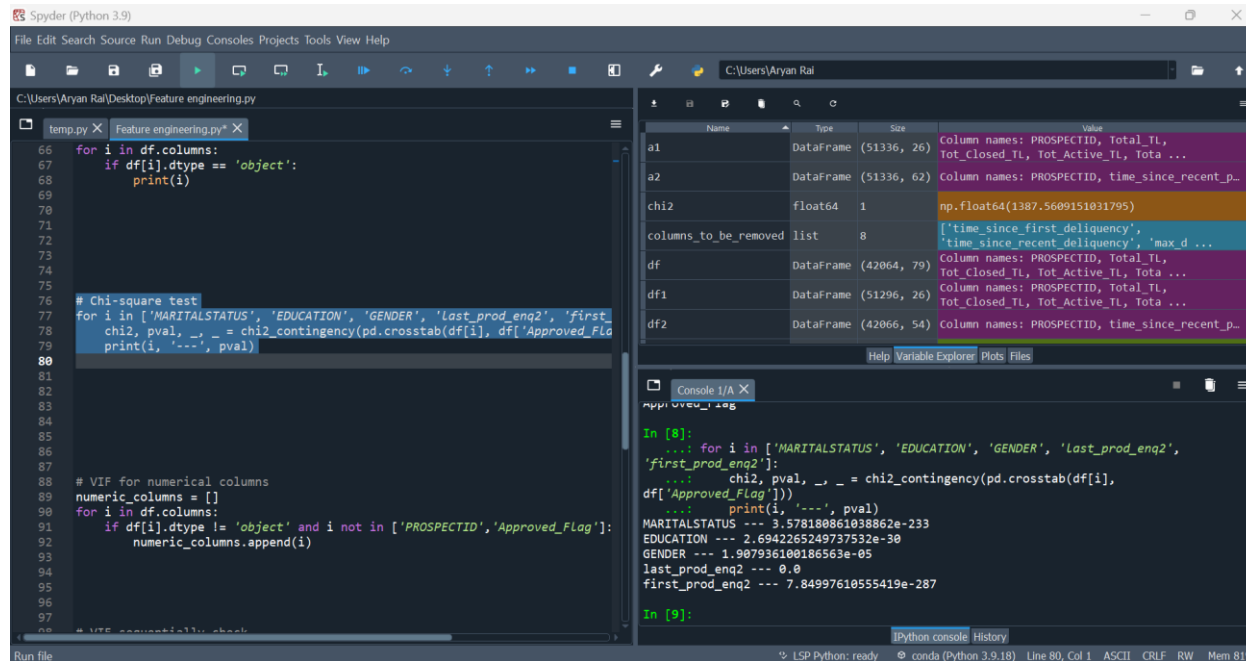
## Example

$$\text{House Price} = (3 \times \text{Floor\_Area}) - (4 \times \text{Distance\_City\_Centre}) - (8 \times \text{House\_Age})$$

$$\text{House Price} = (3 \times \text{Floor\_Area}) - (4 \times \text{Distance\_City\_Centre}) + (7 \times \text{Carpet\_Area})$$

## Chi-Square Test for Categorical Features:

- For each categorical feature (like Marital Status, Education, Gender, etc.), a chi-square test is performed to check if it is associated with the loan approval flag (Approved\_Flag). This test helps determine which categorical features have a significant relationship with the target variable.



The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script for feature engineering and chi-square testing. The script iterates through columns of a DataFrame, identifying categorical features and performing chi-square tests for their association with the 'Approved\_Flag' target variable. The console window at the bottom shows the execution output, including the results of the chi-square tests for 'MARITALSTATUS', 'EDUCATION', 'GENDER', 'last\_prod\_enq2', and 'first\_prod\_enq2'.

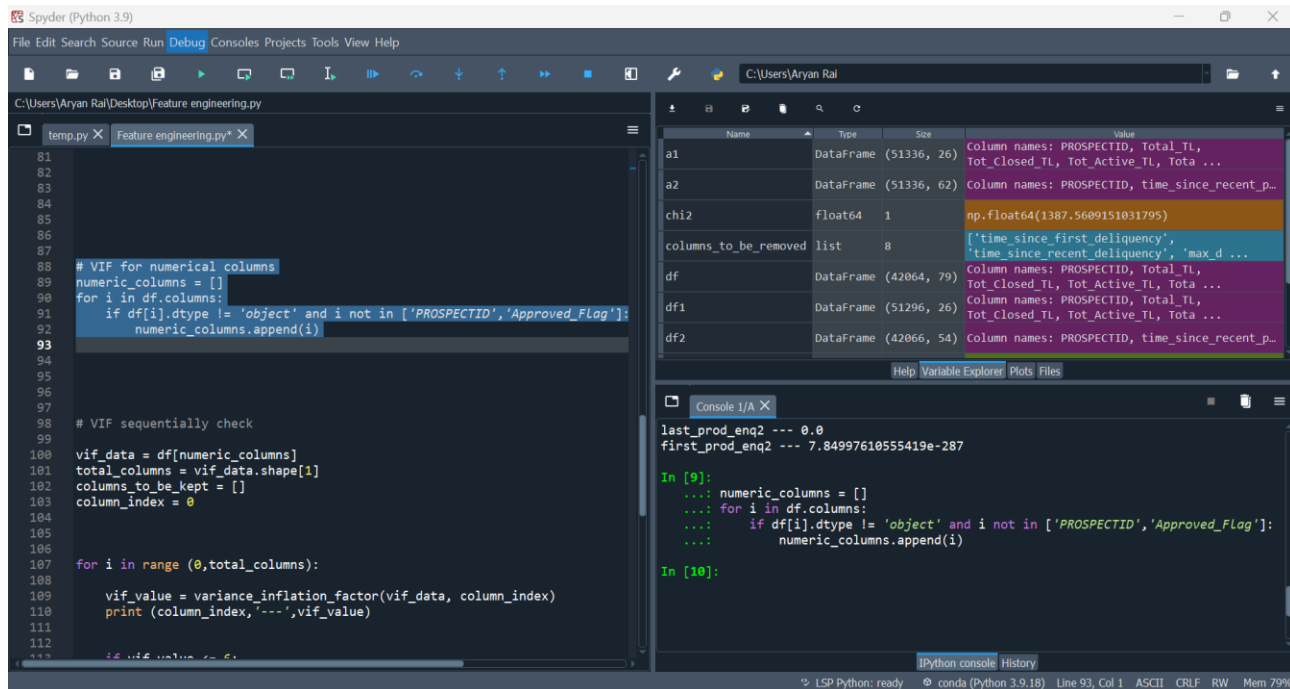
```
temp.py X Feature engineering.py X
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Aryan Rai\Desktop\Feature engineering.py
66 for i in df.columns:
67     if df[i].dtype == 'object':
68         print(i)
69
70
71
72
73
74
75
76 # Chi-square test
77 for i in ['MARITALSTATUS', 'EDUCATION', 'GENDER', 'last_prod_enq2', 'first_
78     chi2, pval, _ = chi2_contingency(pd.crosstab(df[i], df['Approved_Flag']
79     print(i, '---', pval)
80
81
82
83
84
85
86
87
88 # VIF for numerical columns
89 numeric_columns = []
90 for i in df.columns:
91     if df[i].dtype != 'object' and i not in ['PROSPECTID', 'Approved_Flag']:
92         numeric_columns.append(i)
93
94
95
96
97
98 # VIF sequential check
99
```

Name	Type	Size	Value
a1	DataFrame	(51336, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
a2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_p...
chi2	float64	1	np.float64(1387.5609151031795)
columns_to_be_removed	list	8	['time_since_first_delinquency', 'time_since_recent_delinquency', 'max_d...
df	DataFrame	(42064, 79)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
df1	DataFrame	(51296, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
df2	DataFrame	(42066, 54)	Column names: PROSPECTID, time_since_recent_p...

```
Console 1/A X
approved_flag
In [8]:
...: for i in ['MARITALSTATUS', 'EDUCATION', 'GENDER', 'last_prod_enq2',
...:     'first_prod_enq2']:
...:     chi2, pval, _ = 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
In [9]:
Python console History
% LSP Python: ready @ conda (Python 3.9.18) Line 80, Col 1 ASCII CRLF RW Mem 81%
```

## Checking Multicollinearity with VIF:

- For numerical columns, multicollinearity is checked using the Variance Inflation Factor (VIF). Multicollinearity occurs when two or more features are highly correlated, and removing such features can improve the model's accuracy. The code removes any features with a VIF value greater than 6.



The screenshot displays the Spyder Python IDE interface. The left pane shows a Python script named 'temp.py' with the following code:

```
81
82
83
84
85
86
87
88 # VIF for numerical columns
89 numeric_columns = []
90 for i in df.columns:
91     if df[i].dtype != 'object' and i not in ['PROSPECTID', 'Approved_Flag']:
92         numeric_columns.append(i)
93
94
95
96
97
98 # VIF sequentially check
99
100 vif_data = df[numeric_columns]
101 total_columns = vif_data.shape[1]
102 columns_to_be_kept = []
103 column_index = 0
104
105
106
107 for i in range(0, total_columns):
108
109     vif_value = variance_inflation_factor(vif_data, column_index)
110     print(column_index, '---', vif_value)
111
112
113
```

The right pane shows the Variable Explorer with a table of variables:

Name	Type	Size	Value
a1	DataFrame	(51336, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
a2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_p...
chi2	float64	1	np.float64(1387.5609151031795)
columns_to_be_removed	list	8	['time_since_first_delinquency', 'time_since_recent_delinquency', 'max_d...
df	DataFrame	(42064, 79)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
df1	DataFrame	(51296, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot...
df2	DataFrame	(42066, 54)	Column names: PROSPECTID, time_since_recent_p...

Below the Variable Explorer is the IPython console showing the execution of the code:

```
last_prod_enq2 --- 0.0
first_prod_enq2 --- 7.84997610555419e-287

In [9]:
...: numeric_columns = []
...: for i in df.columns:
...:     if df[i].dtype != 'object' and i not in ['PROSPECTID', 'Approved_Flag']:
...:         numeric_columns.append(i)

In [10]:
```

The status bar at the bottom indicates: LSP Python: ready, conda (Python 3.9.18), Line 93, Col 1, ASCII, CRLF, RW, Mem 79%.

Spyder (Python 3.9)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Aryan Rai\Desktop\Feature engineering.py

```
temp.py x Feature engineering.py* x
96
97
98 # VIF sequentially check
99
100 vif_data = df[numeric_columns]
101 total_columns = vif_data.shape[1]
102 columns_to_be_kept = []
103 column_index = 0
104
105
106
107 for i in range(0, total_columns):
108     vif_value = variance_inflation_factor(vif_data, column_index)
109     print(column_index, '-', vif_value)
110
111
112
113     if vif_value <= 6:
114         columns_to_be_kept.append(numeric_columns[i])
115         column_index = column_index + 1
116     else:
117         vif_data = vif_data.drop([numeric_columns[i]], axis=1)
118
119
120
121
122
123
124
125
126 # check Anova for columns_to_be_kept
127
128 from sklearn import metrics
```

C:\Users\Aryan Rai

Name	Type	Size	Value
a1	DataFrame	(51336, 26)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot_...
a2	DataFrame	(51336, 62)	Column names: PROSPECTID, time_since_recent_p...
chi2	float64	1	np.float64(1387.5609151031795)
column_index	int	1	39
columns_to_be_kept	list	39	['pct_tl_open_L6M', 'pct_tl_closed_L6M', 'Tot_TL_closed_L12M', 'pct_tl ...
columns_to_be_removed	list	8	['time_since_first_delinquency', 'time_since_recent_delinquency', 'max_d ...
df	DataFrame	(42064, 79)	Column names: PROSPECTID, Total_TL, Tot_Closed_TL, Tot_Active_TL, Tot_...

Help Variable Explorer Plots Files

Console 1/A x

```
31 --- 3.7247110211057
32 --- 10.175021454450935
32 --- 6.408710354561301
32 --- 1.0011511962625623
33 --- 3.069197305397274
34 --- 2.8091261600643724
35 --- 20.249538381980678
35 --- 15.864576541593745
35 --- 1.8331649740532168
36 --- 1.5680839009542044
37 --- 1.9307572353811677
38 --- 4.331265056645247
39 --- 9.390334396150173

In [11]:
```

Python console History

LSP Python: ready conda (Python 3.9.18) Line 118, Col 66 ASCII CRLF RW Mem 80%



## ANOVA Test for Numerical Features:

- Finally, the code performs an ANOVA test on numerical features to check whether they have a significant relationship with the target variable. Only features with a p-value less than or equal to 0.05 are kept for further analysis.

The screenshot displays the Spyder Python IDE interface. The left pane shows a script named 'Feature engineering.py' with the following code:

```
121
122
123
124
125
126 # check Anova for columns_to_be_kept
127
128 from scipy.stats import f_oneway
129
130 columns_to_be_kept_numerical = []
131
132 for i in columns_to_be_kept:
133     a = list(df[i])
134     b = list(df['Approved_Flag'])
135
136     group_P1 = [value for value, group in zip(a, b) if group == 'P1']
137     group_P2 = [value for value, group in zip(a, b) if group == 'P2']
138     group_P3 = [value for value, group in zip(a, b) if group == 'P3']
139     group_P4 = [value for value, group in zip(a, b) if group == 'P4']
140
141
142     f_statistic, p_value = f_oneway(group_P1, group_P2, group_P3, group_P4)
143
144     if p_value <= 0.05:
145         columns_to_be_kept_numerical.append(i)
146
147
148
149
150
151
152
153
```

The right pane shows the Variable Explorer with the following variables:

Name	Type	Size	Value
i	str	7	GL_Flag
numeric_columns	list	72	['Total_TL', 'Tot_Closed_TL', 'Tot_Act...
p_value	float64	1	np.float64(5e-324)
pval	float64	1	np.float64(7.8499761055419e-287)
total_columns	int	1	72
vif_data	DataFrame	(42064, 39)	column names: pct_tl_open_l6M, pct_tl_...
vif_value	float64	1	np.float64(9.390334396150173)

The bottom pane shows the IPython console with the following output:

```
In [12]:
...: 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_statistic, p_value = f_oneway(group_P1, group_P2, group_P3, group_P4)
...:
...: if p_value <= 0.05:
...:     columns_to_be_kept_numerical.append(i)
...:
```

The status bar at the bottom indicates: LSP Python: ready, conda (Python 3.9.18), Line 145, Col 47, ASCII, CRLF, RW, Mem 80%.

## ➤ Finally we have our single Dataset Ready for Modeling

df - DataFrame

Index	PROSPECTID	Total TI	Tot Closed TI	Tot Active TI	TI opened	TI closed I1	TI open I6	TI closed I6	net active TI	net closed TI	TI opened I1	TI closed I1	TI open I12	TI closed I1	# Missed Pm	Auto TI	CC TI	Consumer TI	Gold TI	Home TI	PI TI
0	1	5	4	1	0	0	0	0	0.2	0.8	0	0	0	0	0	0	0	1	0	4	
1	2	1	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	
2	3	8	0	8	1	0	0.125	0	1	0	2	0	0.25	0	1	1	0	6	1	0	
3	5	3	2	1	0	0	0	0	0.333	0.667	0	0	0	0	0	1	0	0	0	0	
4	6	6	5	1	0	0	0	0	0.167	0.833	0	1	0	0.167	0	4	0	0	2	0	
5	8	6	4	2	0	0	0	0	0.333	0.667	1	2	0.167	0.333	0	1	0	0	0	0	
6	9	1	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	
7	11	7	2	5	1	0	0.143	0	0.714	0.286	3	1	0.429	0.143	0	2	1	2	0	2	
8	13	2	2	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	
9	14	2	1	1	1	0	0.5	0	0.5	0.5	1	1	0.5	0.5	0	0	0	2	0	0	
10	15	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	
11	16	8	1	7	7	1	0.875	0.125	0.875	0.125	8	1	1	0.125	2	1	1	5	0	0	
12	17	1	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	
13	20	7	5	2	1	0	0.143	0	0.286	0.714	1	1	0.143	0.143	0	0	0	0	2	0	
14	21	1	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	
15	22	5	2	3	0	1	0	0.2	0.6	0.4	2	1	0.4	0.2	1	3	0	1	0	1	
16	23	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	
17	25	3	1	2	0	0	0	0	0.667	0.333	1	1	0.333	0.333	0	0	0	0	0	0	
18	27	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	

Format

Resize

☐ Background color

☐ Column min/max

Save and Close

Close

- Here we have done Encoding Categorical Features: Ordinal encoding is applied to the EDUCATION feature. One-hot encoding is applied to categorical variables like MARITALSTATUS, GENDER, and others.

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Aryan Rai

C:\Users\Aryan Rai\Desktop\capstone project work\credit risk assesment using machine learning.py

credit risk assesment using machine learning.py\*

```
154
155
156 # feature selection is done for cat and num features
157
158
159 # listing all the final features
160 features = columns_to_be_kept_numerical + ['MARITALSTATUS', 'EDUCATION', 'GENDER', 'Last_prod_enq2', 'first_prod_enq2']
161 df = df[features + ['Approved_Flag']]
162
163
164
165
166 # Label encoding for the categorical features
167 ['MARITALSTATUS', 'EDUCATION', 'GENDER', 'Last_prod_enq2', 'first_prod_enq2']
168
169
170
171 df['MARITALSTATUS'].unique()
172 df['EDUCATION'].unique()
173 df['GENDER'].unique()
174 df['Last_prod_enq2'].unique()
175 df['first_prod_enq2'].unique()
176
177
178
179 # Ordinal feature -- EDUCATION
180 # SSC : 1
181 # 12TH : 2
182 # GRADUATE : 3
183 # UNDER GRADUATE : 3
184 # POST-GRADUATE : 4
185 # OTHERS : 1
186 # PROFESSIONAL : 3
```

Name	Type	Size
a	list	4200
a1	DataFrame	(513, 18)
a2	DataFrame	(513, 18)
b	list	4200
chi2	float64	1
column_index	int	1

Help Variable Explorer Plots Files

Console 1/A

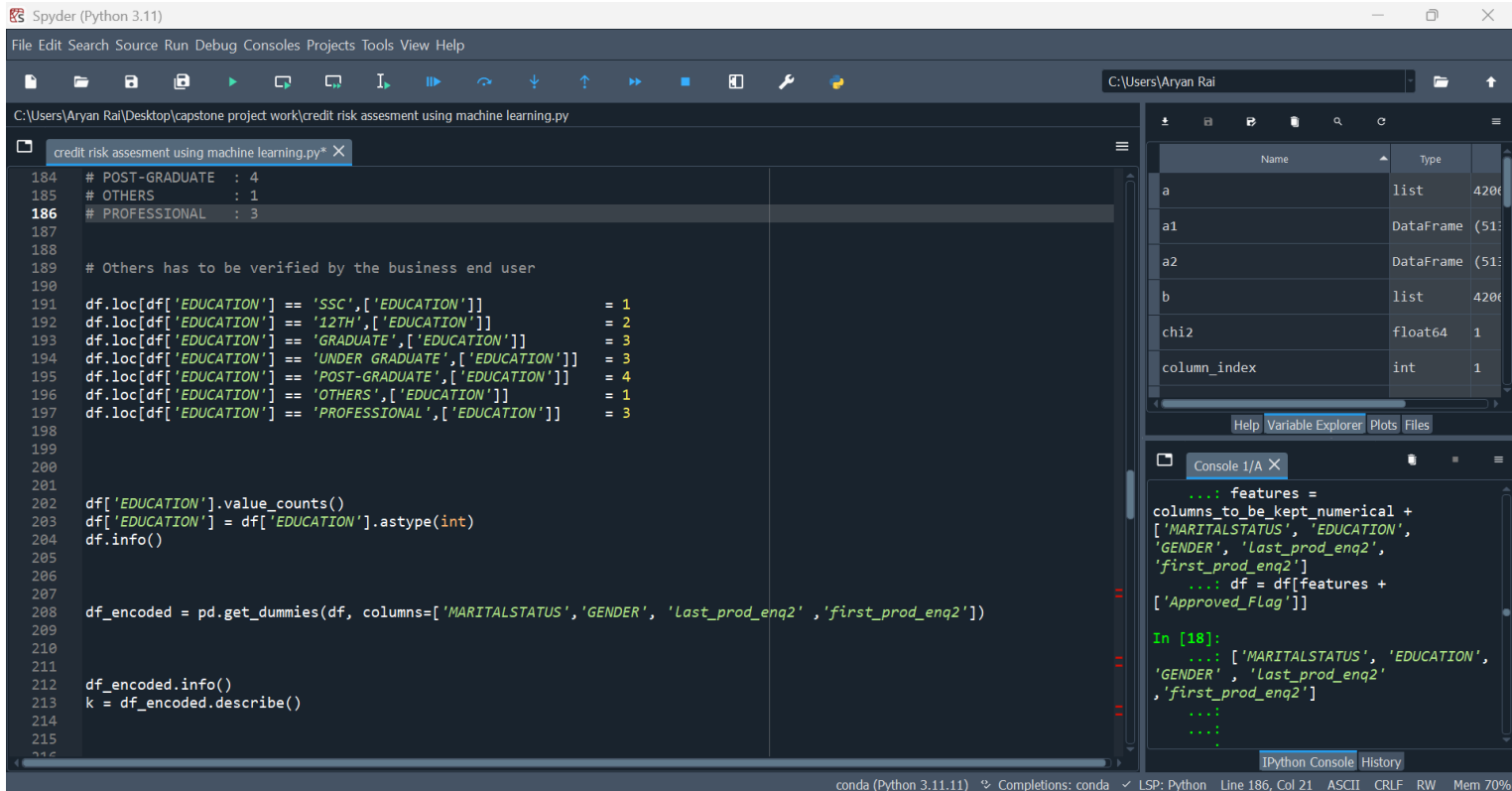
```
...: features =
columns_to_be_kept_numerical +
['MARITALSTATUS', 'EDUCATION',
'GENDER', 'Last_prod_enq2',
'first_prod_enq2']
...: df = df[features +
['Approved_Flag']]

In [18]:
...: ['MARITALSTATUS', 'EDUCATION',
'GENDER', 'Last_prod_enq2',
'first_prod_enq2']
...:
...:
```

IPython Console History

conda (Python 3.11.1) % Completions: conda < LSP: Python Line 186 Col 21 ASCII CR LF RW Mem 70%

- Then we have described the data for checking the data type after label encoding and also we have checked that our target variable count after label encoding to check for dataset is balanced or imbalanced



conda (Python 3.11.11) File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Aryan Rai\Desktop\capstone project work\credit risk assesment using machine learning.py

```
184 # POST-GRADUATE : 4
185 # OTHERS : 1
186 # PROFESSIONAL : 3
187
188 # Others has to be verified by the business end user
189
190 df.loc[df['EDUCATION'] == 'SSC', ['EDUCATION']] = 1
191 df.loc[df['EDUCATION'] == '12TH', ['EDUCATION']] = 2
192 df.loc[df['EDUCATION'] == 'GRADUATE', ['EDUCATION']] = 3
193 df.loc[df['EDUCATION'] == 'UNDER GRADUATE', ['EDUCATION']] = 3
194 df.loc[df['EDUCATION'] == 'POST-GRADUATE', ['EDUCATION']] = 4
195 df.loc[df['EDUCATION'] == 'OTHERS', ['EDUCATION']] = 1
196 df.loc[df['EDUCATION'] == 'PROFESSIONAL', ['EDUCATION']] = 3
197
198 df['EDUCATION'].value_counts()
199 df['EDUCATION'] = df['EDUCATION'].astype(int)
200 df.info()
201
202 df_encoded = pd.get_dummies(df, columns=['MARITALSTATUS', 'GENDER', 'Last_prod_enq2', 'first_prod_enq2'])
203
204 df_encoded.info()
205 k = df_encoded.describe()
206
```

Name	Type	Count
a	list	4206
a1	DataFrame	(51)
a2	DataFrame	(51)
b	list	4206
chi2	float64	1
column_index	int	1

Help Variable Explorer Plots Files

Console 1/A X

```
...: features =
columns_to_be_kept_numerical +
['MARITALSTATUS', 'EDUCATION',
'GENDER', 'Last_prod_enq2',
'first_prod_enq2']
...: df = df[features +
['Approved_Flag']]

In [18]:
...: ['MARITALSTATUS', 'EDUCATION',
'GENDER', 'Last_prod_enq2',
'first_prod_enq2']
...:
...:
```

IPython Console History

conda (Python 3.11.11) Completions: conda LSP: Python Line 186, Col 21 ASCII CRLF RW Mem 70%

# Confusion Matrix

Accuracy, Recall, Precision, F1 score

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	<b>Sensitivity</b> $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	<b>Specificity</b> $\frac{TN}{(TN + FP)}$
		<b>Precision</b> $\frac{TP}{(TP + FP)}$	<b>Negative Predictive Value</b> $\frac{TN}{(TN + FN)}$	<b>Accuracy</b> $\frac{TP + TN}{(TP + TN + FP + FN)}$

$$F1 \text{ score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- Applied Random forest Model

Spyder (Python 3.11)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Aryan Rai\Desktop\capstone project work\credit risk assesment using machine learning.py

```
credit risk assesment using machine learning.py* X
231
232 # Data processing
233
234 # 1. Random Forest
235
236 y = df_encoded['Approved_Flag']
237 x = df_encoded.drop ( ['Approved_Flag'], axis = 1 )
238
239 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
240
241 rf_classifier = RandomForestClassifier(n_estimators = 200, random_state=42)
242
243 rf_classifier.fit(x_train, y_train)
244
245 y_pred = rf_classifier.predict(x_test)
246
247 accuracy = accuracy_score(y_test, y_pred)
248 print ()
249 print(f'Accuracy: {accuracy}')
250 print ()
251 precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
252
253
254 for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
255     print(f"Class {v}:")
256     print(f"Precision: {precision[i]}")
257     print(f"Recall: {recall[i]}")
258     print(f"F1 Score: {f1_score[i]}")
259     print()
260
261
262
263
```

C:\Users\Aryan Rai

Name	Type
k	DataFrame

Help Variable Explorer Plots Files

Console 1/A X

```
...: print()
Accuracy: 0.7636990372043266

Class p1:
Precision: 0.8370457209847597
Recall: 0.7041420118343196
F1 Score: 0.7648634172469202

Class p2:
Precision: 0.7957519116397621
Recall: 0.9282457879088206
F1 Score: 0.856907593778591

Class p3:
Precision: 0.4423380726698262
Recall: 0.21132075471698114
F1 Score: 0.28600612870275793

Class p4:
Precision: 0.7178502879078695
Recall: 0.7269193391642371
F1 Score: 0.7223563495895703
```

IPython Console History

- Applied Xgboost model

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script for training an XGBoost classifier. The script includes imports for XGBoost and LabelEncoder, followed by data preprocessing steps like encoding and splitting. The XGBoost classifier is trained on the training data, and its performance is evaluated on the test data using accuracy, precision, recall, and F1 score. The results are printed to the console.

```
266 # 2. xgboost
267
268 import xgboost as xgb
269 from sklearn.preprocessing import LabelEncoder
270
271 xgb_classifier = xgb.XGBClassifier(objective='multi:softmax', num_class=4)
272
273 y = df_encoded['Approved_Flag']
274 x = df_encoded.drop ( ['Approved_Flag'], axis = 1 )
275
276
277 label_encoder = LabelEncoder()
278 y_encoded = label_encoder.fit_transform(y)
279
280 x_train, x_test, y_train, y_test = train_test_split(x, y_encoded, test_size=0.2, random_state=42)
281
282 xgb_classifier.fit(x_train, y_train)
283 y_pred = xgb_classifier.predict(x_test)
284
285 accuracy = accuracy_score(y_test, y_pred)
286 print ()
287 print(f'Accuracy: {accuracy:.2f}')
288 print ()
289
290 precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
291
292 for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
293     print(f"Class {v}:")
294     print(f"Precision: {precision[i]}")
295     print(f"Recall: {recall[i]}")
296     print(f"F1 Score: {f1_score[i]}")
297     print()
```

The Variable Explorer on the right shows a variable 'k' of type 'DataFrame'. The Console window displays the following output:

```
Accuracy: 0.78

Class p1:
Precision: 0.823906083244397
Recall: 0.7613412228796844
F1 Score: 0.7913890312660175

Class p2:
Precision: 0.8255418233924413
Recall: 0.913577799801784
F1 Score: 0.8673315769665035

Class p3:
Precision: 0.4756380510440835
Recall: 0.30943396226415093
F1 Score: 0.37494284407864653

Class p4:
Precision: 0.7342386032977691
Recall: 0.7356656948493683
F1 Score: 0.7349514563106796

In [22]:
```

The status bar at the bottom indicates the environment is conda (Python 3.11.11) with 72% memory usage.

- Applied Decision Tree

The screenshot displays the Spyder Python IDE (Python 3.11) with a file named 'credit risk assesment using machine learning.py' open. The code implements a Decision Tree Classifier using sklearn. The model is trained on a dataset with 'Approved\_Flag' as the target variable. The performance metrics for four classes (p1, p2, p3, p4) are displayed in the IPython Console.

```
300
301
302
303 # 3. Decision Tree
304 from sklearn.tree import DecisionTreeClassifier
305
306
307 y = df_encoded['Approved_Flag']
308 x = df_encoded.drop ( ['Approved_Flag'], axis = 1 )
309
310 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
311
312
313 dt_model = DecisionTreeClassifier(max_depth=20, min_samples_split=10)
314 dt_model.fit(x_train, y_train)
315 y_pred = dt_model.predict(x_test)
316
317 accuracy = accuracy_score(y_test, y_pred)
318 print ()
319 print(f"Accuracy: {accuracy:.2f}")
320 print ()
321
322 precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
323
324 for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
325     print(f"Class {v}:")
326     print(f"Precision: {precision[i]}")
327     print(f"Recall: {recall[i]}")
328     print(f"F1 Score: {f1_score[i]}")
329     print()
330
331
332
```

The IPython Console shows the following output:

```
Accuracy: 0.71

Class p1:
Precision: 0.7211350293542075
Recall: 0.7268244575936884
F1 Score: 0.7239685658153242

Class p2:
Precision: 0.8102743724460012
Recall: 0.8253716551040634
F1 Score: 0.8177533385703064

Class p3:
Precision: 0.3462757527733756
Recall: 0.329811320754717
F1 Score: 0.3378430614611519

Class p4:
Precision: 0.6525252525252525
Recall: 0.6277939747327502
F1 Score: 0.6399207528479445

In [23]:
```



**Thank You**