Sri Lanka Institute of Information Technology

Fundamentals Of Data Mining (IT3051)

Continuous Assignment – 2024, Semester 1

Final Report – Mini Project

Ranasinghe R.Y.G	IT22253880
Gamage M.P.L	IT22578082
Thiyanima H.E.S	IT22271600
Dilshan H.M.T.W	IT22562456
Handapangoda C.N	IT22586070

Table of Content

Introduction

In the ever-evolving financial landscape, vehicle loan companies face increasing pressure to make informed, data-driven decisions about potential borrowers. The challenge lies in accurately assessing the risk associated with each loan applicant, ensuring that loans are granted to clients who are likely to repay, thereby reducing defaults and financial losses. To address this challenge, we are developing a predictive system using advanced machine learning techniques that helps vehicle loan companies determine whether a client is suitable or not for receiving a vehicle loan.

This project involves the use of historical data related to clients' financial and personal attributes to train a predictive model capable of classifying applicants based on their repayment likelihood. By leveraging client variables such as income, credit history, employment status, and existing liabilities, the model aims to provide a reliable assessment of loan repayment capability.

The demand for predictive solutions in the financial sector is on the rise, driven by the need for minimizing loan defaults while maximizing profitable lending opportunities. Traditional methods of credit assessment, such as manual review and scoring, are often time-consuming and prone to inaccuracies due to human error and bias. In contrast, automated machine learning models provide a scalable, unbiased, and data-driven approach to evaluating creditworthiness, enhancing the efficiency and effectiveness of decision-making.

Moreover, the rapid growth in the vehicle loan market, fueled by the increased affordability of vehicles and favorable lending conditions, has led to a surge in demand for more sophisticated risk management tools. Predictive systems not only benefit financial institutions by reducing risks but also they provide a smoother and faster experience for borrowers, who can receive timely responses to their loan applications. This project aims to fulfill these needs by empowering vehicle loan companies with a robust solution to enhance their decision-making processes in today's competitive and dynamic market.

Dataset Features

Dataset – Automobile Loan Default Dataset (kaggle.com)

Features	Description
Client_Income	Income of the client
Bike_Owned	Number of bikes owned by the client
Active_Loan	Number of active loans
House_Own	Number of houses owned by the client
Child_Count	Number of children client has
Credit_Amount	Loan amount requested by the applicant
Loan_Annuity	Periodic loan repayment amount
Accompany_Client	Client accompanied by another person.
Client_Income_Type	Income type of the client
Client_Education	Education level of the client
Client_Marital_Status	Married or unmarried
Client_Gender	Gender of the client
Loan_Contract_Type	Type of the loan
Client_Housing_Type	Type of housing client resides in.
Population_Region_Relative	Comparison of population size by region
Employed_Days	Employed days of the client
Registration_Days	Registration days
Own_House_Age	Age of the client’s own house
Workphone_Working	Whether the work phone working or not
Client_Occupation	Job or profession of the client
Client_Family_Members	Number of family members

Cleint_City_Rating	Rating of the clients residential city
Application_Process_Day	Application process day
Application_Process_Hour	Application process hour
Type_Organization	Organization type
Credit_Bureau	Agency that collects credit information

Data Collection

- Importing Packages

```
[1]: import pandas as pd
     import numpy as np
     import pickle
```

1.Data Collection

```
[2]: df = pd.read_csv("C:\\Users\\User\\Desktop\\Dataset\\Automobile_Loan.csv",low_memory=False)
```

```
[3]: #first few rows of dataset
     df.head()
```

[3]:

	ID	Client_Income	Car_Owned	Bike_Owned	Active_Loan	House_Own	Child_Count	Credit_Amount	Loan_Annuity	Accompany_Client	...	Client_Permanent_M
0	12142509	6750	0.0	0.0	1.0	0.0	0.0	61190.55	3416.85	Alone	...	
1	12138936	20250	1.0	0.0	1.0	NaN	0.0	15282	1826.55	Alone	...	
2	12181264	18000	0.0	0.0	1.0	0.0	1.0	59527.35	2788.2	Alone	...	
3	12188929	15750	0.0	0.0	1.0	1.0	0.0	53870.4	2295.45	Alone	...	
4	12133385	33750	1.0	0.0	1.0	0.0	2.0	133988.4	3547.35	Alone	...	

5 rows × 40 columns

Feature Engineering

```
[4]: #check dataset's info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 121856 entries, 0 to 121855
Data columns (total 40 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   ID                          121856 non-null  int64
 1   Client_Income               118249 non-null  object
 2   Car_Owned                   118275 non-null  float64
 3   Bike_Owned                  118232 non-null  float64
 4   Active_Loan                 118221 non-null  float64
 5   House_Own                   118195 non-null  float64
 6   Child_Count                 118218 non-null  float64
 7   Credit_Amount               118224 non-null  object
 8   Loan_Annuity                117044 non-null  object
 9   Accompany_Client            120110 non-null  object
 10  Client_Income_Type          118155 non-null  object
 11  Client_Education            118211 non-null  object
 12  Client_Marital_Status       118383 non-null  object
 13  Client_Gender               119443 non-null  object
 14  Loan_Contract_Type          118205 non-null  object
 15  Client_Housing_Type         118169 non-null  object
 16  Population_Region_Relative  116999 non-null  object
 17  Age_Days                    118256 non-null  object
 18  Employed_Days               118207 non-null  object
 19  Registration_Days           118242 non-null  object
 20  ID_Days                     115888 non-null  object
 21  Own_House_Age               41761 non-null   float64
 22  Mobile_Tag                  121856 non-null  int64
 23  Homephone_Tag               121856 non-null  int64
 24  Workphone_Working           121856 non-null  int64
 25  Client_Occupation           80421 non-null   object
 26  Client_Family_Members       119446 non-null  float64
 27  Cleint_City_Rating          119447 non-null  float64
 28  Application_Process_Day     119428 non-null  float64
 29  Application_Process_Hour    118193 non-null  float64
 30  Client_Permanent_Match_Tag  121856 non-null  object
 31  Client_Contact_Work_Tag     121856 non-null  object
 32  Type_Organization           118247 non-null  object
 33  Score_Source_1              53021 non-null   float64
 34  Score_Source_2              116170 non-null  float64
 35  Score_Source_3              94935 non-null   object
 36  Social_Circle_Default       59928 non-null   float64
 37  Phone_Change                118192 non-null  float64
 38  Credit_Bureau               103316 non-null  float64
 39  Default                     121856 non-null  int64
dtypes: float64(15), int64(5), object(20)
memory usage: 37.2+ MB
```

```
[6]: #drop [id] columns because it's not necessary
df.drop(columns=['ID', 'Client_Permanent_Match_Tag', 'Client_Contact_Work_Tag','Car_Owned','Mobile_Tag'], inplace=True)
```

Data Cleaning

[10]:

```
#check the missing value
df.isna().sum().sort_values(ascending=False)
```

[10]:

```
Own_House_Age                 80095
Score_Source_1                68835
Social_Circle_Default         61928
Client_Occupation             41435
Score_Source_3                26922
Credit_Bureau                 18540
ID_Days                        5985
Score_Source_2                 5686
Population_Region_Relative     4868
Loan_Annuity                   4826
Client_Income_Type             3701
Client_Housing_Type            3687
Employed_Days                  3666
Phone_Change                   3664
Application_Process_Hour       3663
House_Own                      3661
Loan_Contract_Type             3651
Client_Education               3645
Child_Count                    3638
Credit_Amount                  3637
Active_Loan                    3635
Registration_Days              3631
Bike_Owned                     3624
Client_Income                  3622
Age_Days                       3617
Type_Organization              3609
Client_Marital_Status          3473
Application_Process_Day        2428
Client_Gender                  2413
Client_Family_Members          2410
Cleint_City_Rating             2409
Accompany_Client               1746
Workphone_Working                 0
Homephone_Tag                     0
Default                           0
dtype: int64
```

[11]:

```
#Remove all null values
df.dropna(axis=0, inplace=True)
df
```

[11]:

	Client_Income	Bike_Owned	Active_Loan	House_Own	Child_Count	Credit_Amount	Loan_Annuity	Accompany_Client	Client_Income_Type	Client_Education	...
12	27000.0	0.0	0.0	1.0	0.0	28440.00	1913.40	Alone	Service	Secondary	...
102	27000.0	0.0	0.0	1.0	3.0	53366.85	4003.20	Alone	Commercial	Secondary	...
126	18000.0	0.0	1.0	1.0	1.0	27000.00	724.95	Alone	Service	Secondary	...
161	18000.0	1.0	1.0	1.0	0.0	48149.55	3351.15	Alone	Service	Secondary	...
189	15750.0	1.0	0.0	1.0	0.0	57340.80	2072.70	Alone	Service	Graduation	...
...	...	...	...	...	...	...	...	...	...	...	...
121684	9000.0	0.0	0.0	0.0	1.0	32590.80	1663.65	Alone	Service	Secondary	...
121771	16650.0	0.0	0.0	1.0	0.0	51244.65	3741.75	Alone	Govt Job	Secondary	...
121777	9900.0	1.0	1.0	1.0	1.0	76022.55	3028.05	Alone	Service	Secondary	...
121838	31500.0	0.0	0.0	1.0	1.0	94230.00	2767.95	Alone	Govt Job	Secondary	...
121854	38250.0	1.0	0.0	1.0	0.0	45000.00	2719.35	Alone	Service	Graduation	...

2633 rows × 35 columns

```
[12]: #check the missing value
      df.isna().sum().sort_values(ascending=False)
```

```
[12]: Client_Income                 0
      Application_Process_Hour      0
      Homephone_Tag                 0
      Workphone_Working             0
      Client_Occupation             0
      Client_Family_Members         0
      Cleint_City_Rating            0
      Application_Process_Day       0
      Type_Organization             0
      ID_Days                       0
      Score_Source_1                0
      Score_Source_2                0
      Score_Source_3                0
      Social_Circle_Default         0
      Phone_Change                  0
      Credit_Bureau                 0
      Own_House_Age                 0
      Registration_Days             0
      Bike_Owned                    0
      Client_Income_Type            0
      Active_Loan                   0
      House_Own                     0
      Child_Count                   0
      Credit_Amount                 0
      Loan_Annuity                  0
      Accompany_Client              0
      Client_Education              0
      Employed_Days                 0
      Client_Marital_Status         0
      Client_Gender                 0
      Loan_Contract_Type            0
      Client_Housing_Type           0
      Population_Region_Relative    0
      Age_Days                      0
      Default                       0
      dtype: int64
```

```
[14]: #check for duplicate values
      df.duplicated().sum()
```

```
[14]: 129
```

```
[15]: #Drop duplicates
      df = df.drop_duplicates()
```

```
[16]: #check for duplicate values after removing them
      df.duplicated().sum()
```

```
[16]: 0
```

```
[88]: (df == 0).sum()
```

```
[88]:  no_of_dependents            712
       education                     0
       self_employed                 0
       income_annum                  0
       loan_amount                   0
       loan_term                     0
       cibil_score                   0
       residential_assets_value      0
       commercial_assets_value       0
       luxury_assets_value           0
       bank_asset_value              0
       loan_status                   0
      dtype: int64
```

```
[18]: #Encode Categorical Variables

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

for column in Cat_val:
    if column in df_encoded.columns:  # Check if the column exists
        df_encoded[column] = label_encoder.fit_transform(df_encoded[column].astype(str))
    else:
        print(f"Warning: {column} does not exist in the DataFrame")
```

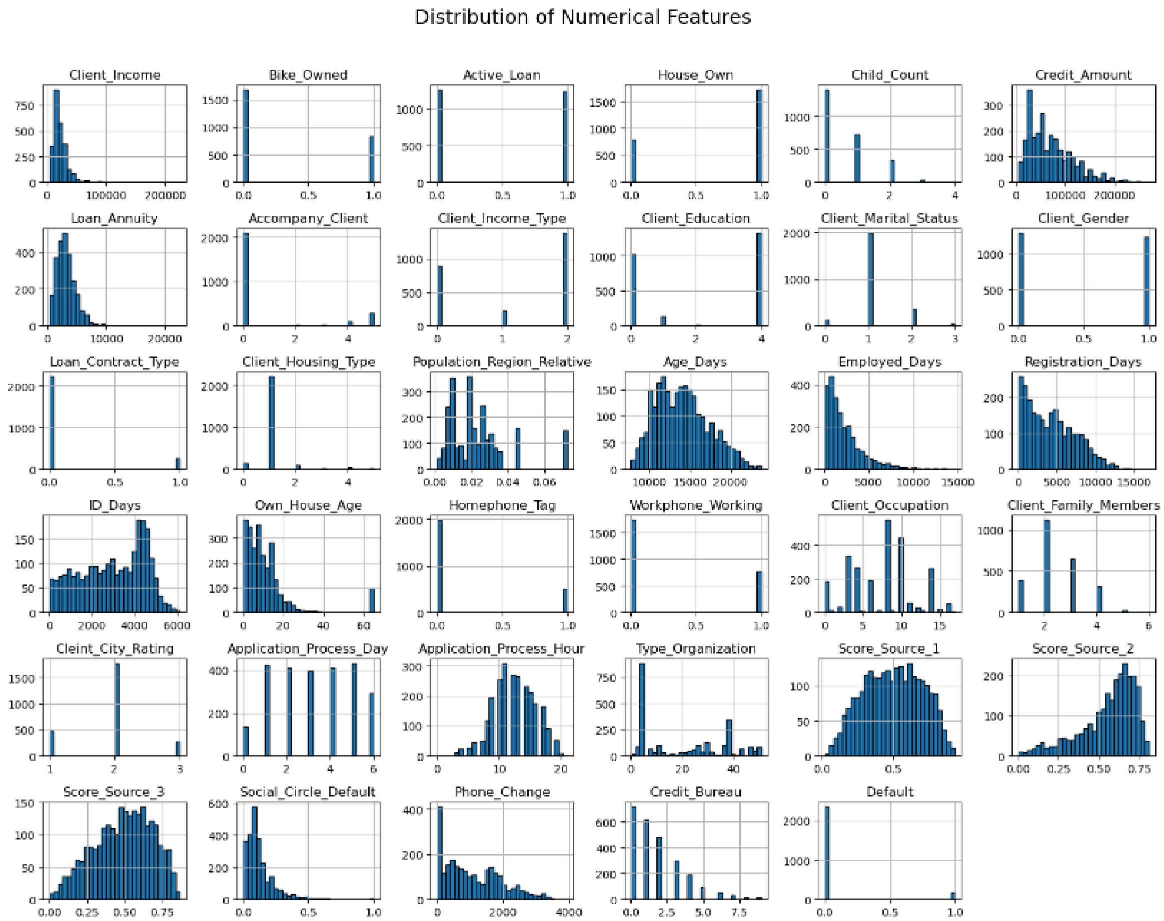
```
[19]: print(df_encoded.head())
```

```
     Client_Income  Bike_Owned  Active_Loan  House_Own  Child_Count  \
12         27000.0         0.0          0.0        1.0          0.0
102        27000.0         0.0          0.0        1.0          3.0
126        18000.0         0.0          1.0        1.0          1.0
161        18000.0         1.0          1.0        1.0          0.0
189        15750.0         1.0          0.0        1.0          0.0

     Credit_Amount  Loan_Annuity  Accompany_Client  Client_Income_Type  \
12        28440.00       1913.40                 0                   2
102       53366.85       4003.20                 0                   0
126       27000.00        724.95                 0                   2
161       48149.55       3351.15                 0                   2
189       57340.80       2072.70                 0                   2

     Client_Education  ...  Application_Process_Day  Application_Process_Hour  \
12                  4  ...                      4.0                      13.0
102                 4  ...                      4.0                      10.0
126                 4  ...                      2.0                      11.0
161                 4  ...                      6.0                      11.0
189                 0  ...                      0.0                      10.0

     Type_Organization  Score_Source_1  Score_Source_2  Score_Source_3  \
12                  39        0.268014        0.684114        0.493863
102                  5        0.477169        0.677447        0.581484
126                 50        0.741930        0.642445        0.397946
161                 39        0.135435        0.470134        0.236611
189                  7        0.288840        0.272040        0.684828

     Social_Circle_Default  Phone_Change  Credit_Bureau  Default
12                  0.1485           0.0            6.0        0
102                 0.0330        1805.0            4.0        0
126                 0.1010        2268.0            0.0        0
161                 0.0021        1753.0            7.0        0
189                 0.0412        1198.0            3.0        0

[5 rows x 35 columns]
```

Exploratory Data Analysis

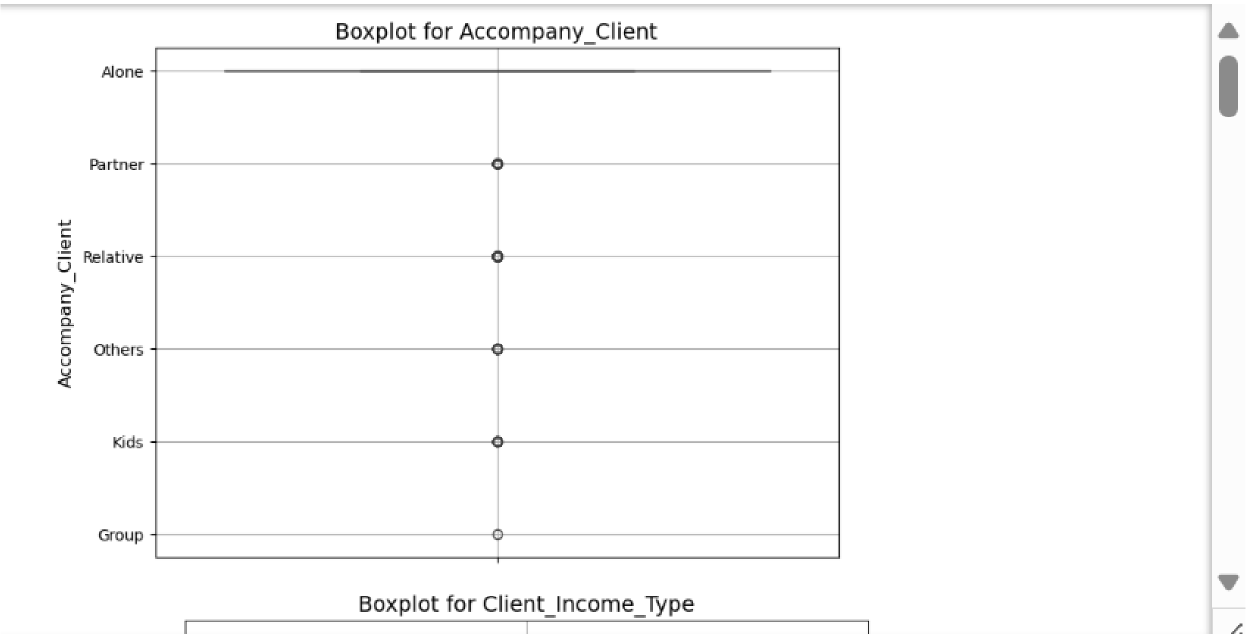
[20]:

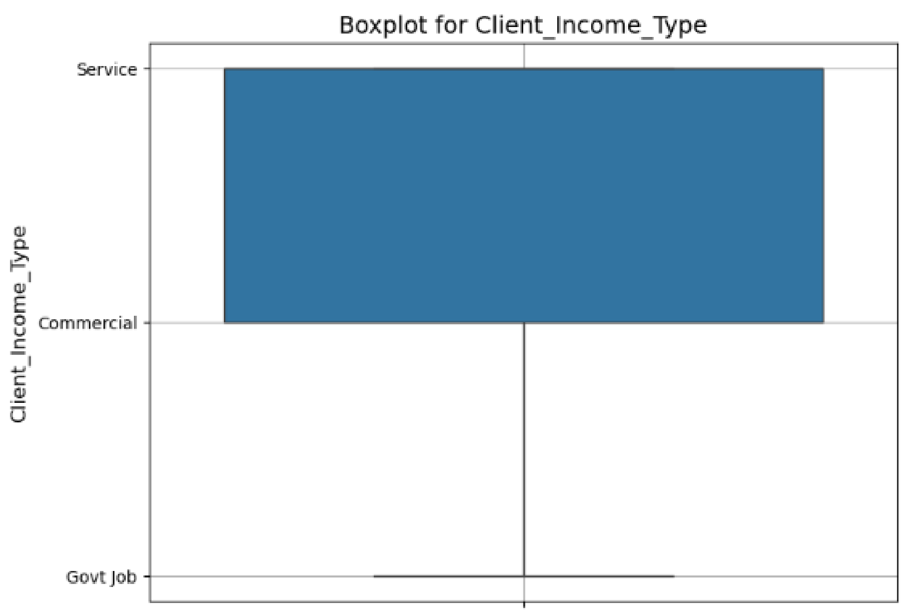
```
#Distribution of Numerical Features (Histograms)

import matplotlib.pyplot as plt

# Plot histograms for numerical features
df_encoded.hist(figsize=(15, 12), bins=30, edgecolor='black')
plt.suptitle('Distribution of Numerical Features', fontsize=18)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

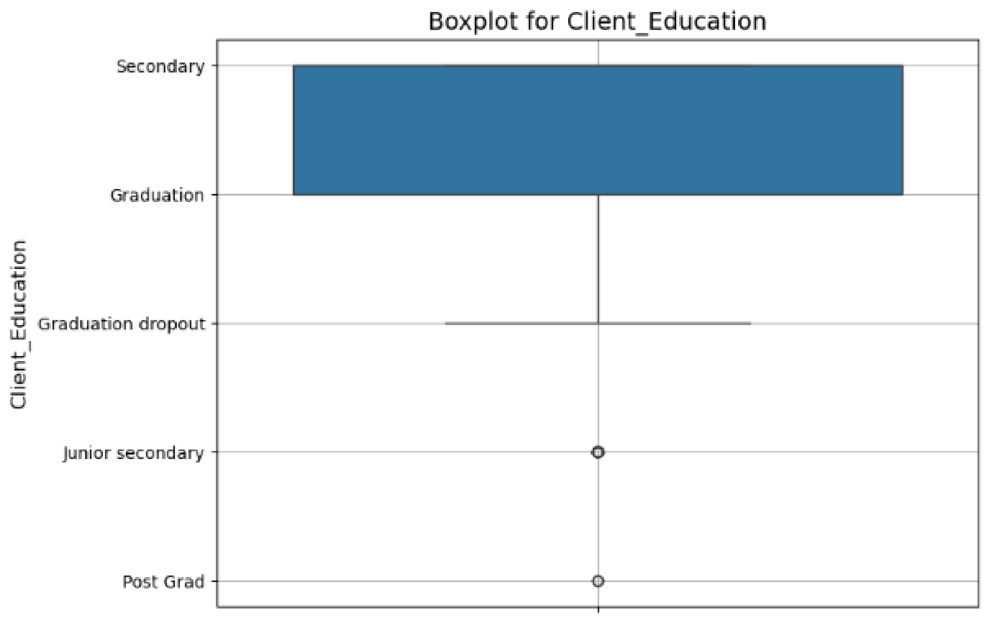


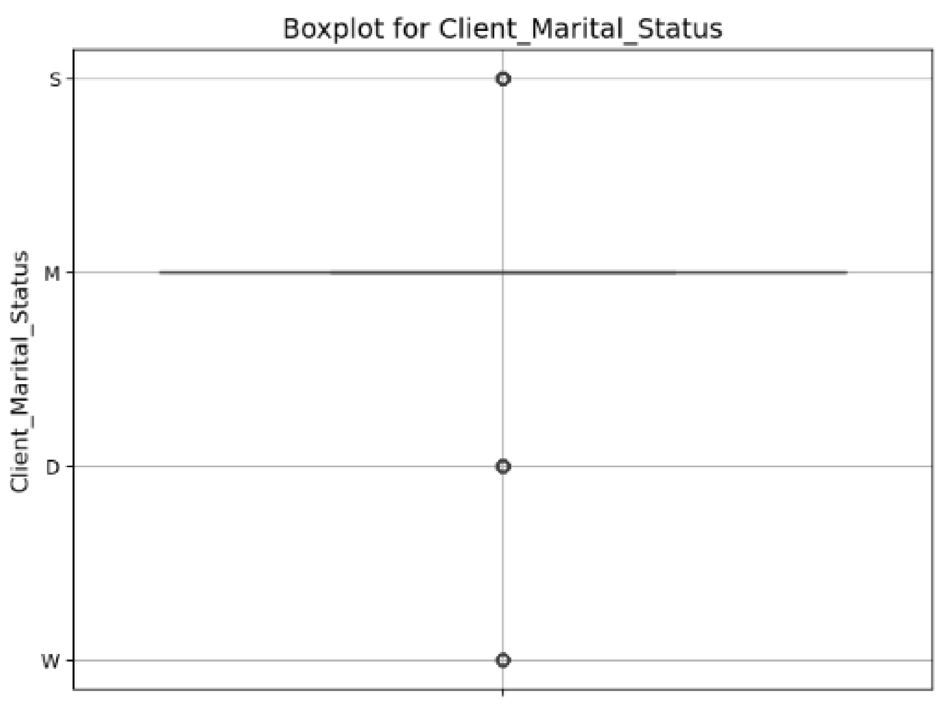
[21]:

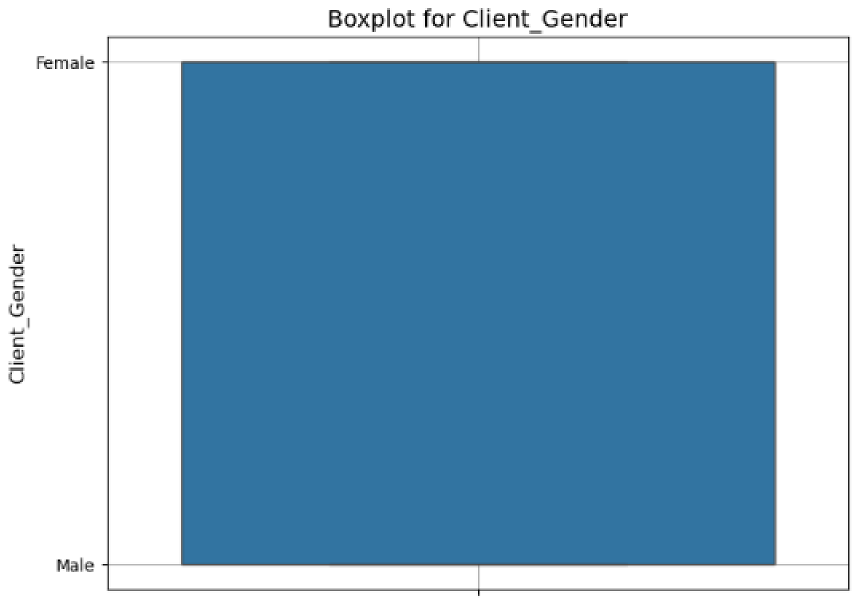
```
# Loop through each selected column to plot a boxplot

import seaborn as sns
import matplotlib.pyplot as plt

for feature in Cat_val:
    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df, y=feature)  # Plot boxplot
    plt.title(f'Boxplot for {feature}', fontsize=14)  # Title
    plt.ylabel(feature, fontsize=12)  # Label for y-axis
    plt.xlabel('')  # Clear x-axis label as it's not needed
    plt.grid(True)  # Add grid for better visualization
    plt.show()  # Display the plot
```

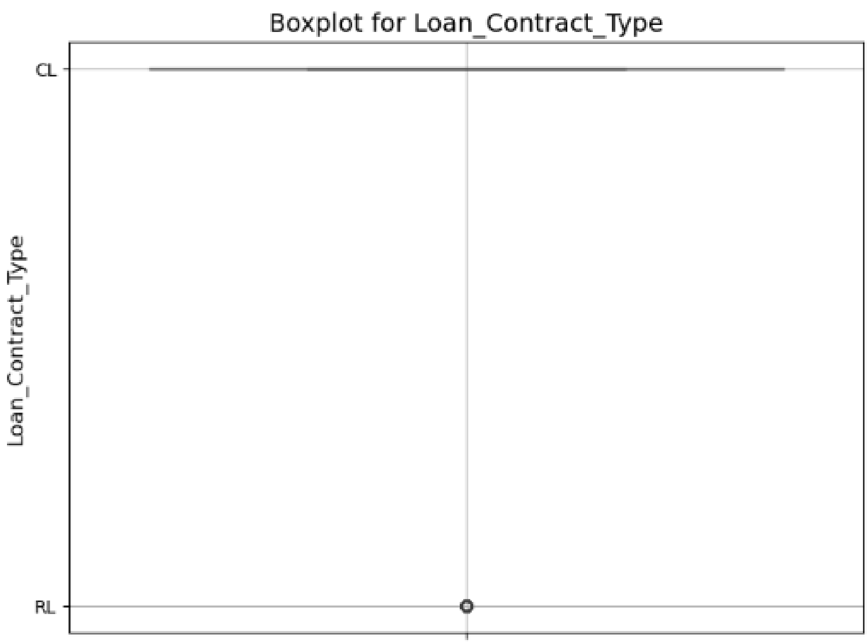




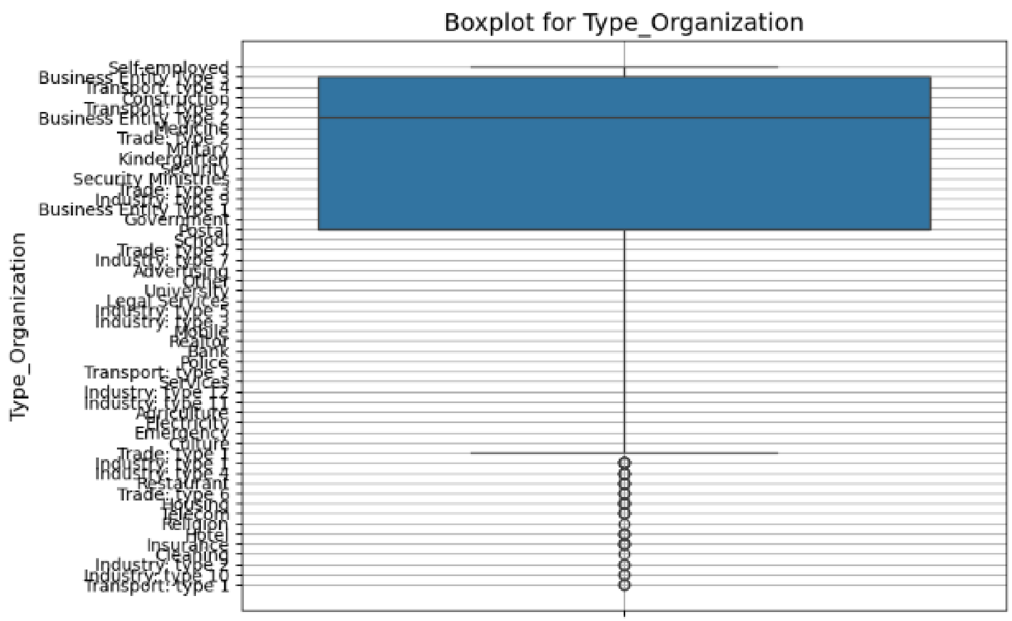

Boxplot for Client_Education
Secondary
Graduation
Graduation dropout
Junior secondary
Post Grad
Client_Education

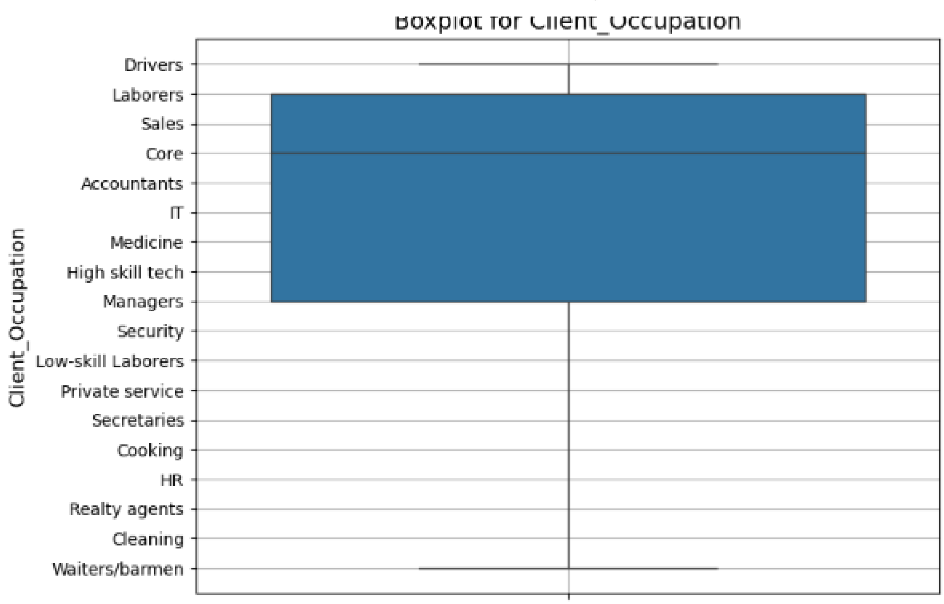

Boxplot for Client_Marital_Status
S
M
D
W
Client_Marital_Status


Boxplot for Client_Gender
Female
Male
Client_Gender


Boxplot for Loan_Contract_Type
CL
RL
Loan_Contract_Type

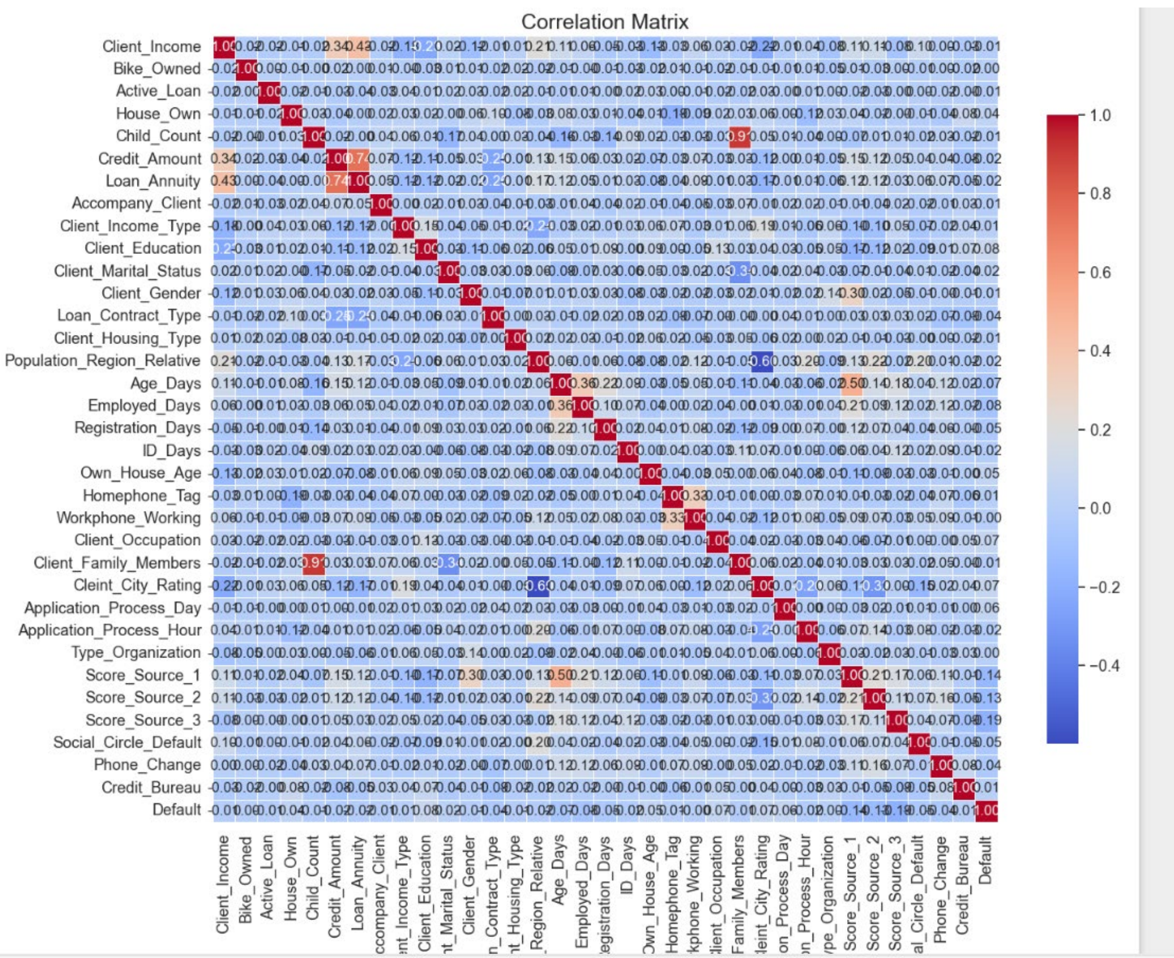

Boxplot for Client_Housing_Type
Client_Housing_Type
Family
Home
Rental
Shared
Office
Municipal


Boxplot for Type_Organization
Type_Organization


Boxplot for Client_Occupation
Client_Occupation
Drivers
Laborers
Sales
Core
Accountants
IT
Medicine
High skill tech
Managers
Security
Low-skill Laborers
Private service
Secretaries
Cooking
HR
Realty agents
Cleaning
Waiters/barmen

[25]:

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

# Compute the correlation matrix
corr = df_encoded.corr()

# Set up the matplotlib figure
plt.figure(figsize=(15, 12))  # Increase the figure size

# Adjust the font size for better readability
sns.set(font_scale=1.2)

# Draw the heatmap
heatmap = sns.heatmap(
    corr,
    annot=True,
    fmt='.2f',
    cmap='coolwarm',
    square=True,
    cbar_kws={"shrink": .8},
    linewidths=0.5,  # Add space between squares for clarity
    annot_kws={"size": 12}  # Increase the annotation font size
)

# Set title
plt.title('Correlation Matrix', fontsize=18)
plt.show()
```



Data Preprocessing

```
[24]: #Outliers detection
      from scipy.stats import zscore

      #create a copy of the Dataframe to avoid modifying the original
      df_copy= df_balanced.copy()

      #Calculate Z-scores for each numeric column
      numeric_columns = df_balanced.select_dtypes(include=[np.number]).columns
      df_balanced[numeric_columns] = df_balanced[numeric_columns].apply(zscore)

      #set a threshold for Z-score
      threshold = 3

      #Identify outliers based on Z-score
      outliers = df_balanced[(np.abs(df_copy[numeric_columns]) > threshold).any(axis=1)]

      print(outliers.count())
```

```
Client_Income                  4696
Bike_Owned                     4696
Active_Loan                    4696
House_Own                      4696
Child_Count                    4696
Credit_Amount                  4696
Loan_Annuity                   4696
Accompany_Client               4696
Client_Income_Type             4696
Client_Education               4696
Client_Marital_Status          4696
Client_Gender                  4696
Loan_Contract_Type             4696
Client_Housing_Type            4696
Population_Region_Relative     4696
Age_Days                       4696
Employed_Days                  4696
Registration_Days              4696
ID_Days                        4696
Own_House_Age                  4696
Homephone_Tag                  4696
Workphone_Working              4696
Client_Occupation              4696
Client_Family_Members          4696
Cleint_City_Rating             4696
Application_Process_Day        4696
Application_Process_Hour       4696
Type_Organization              4696
Score_Source_1                 4696
Score_Source_2                 4696
Score_Source_3                 4696
Social_Circle_Default          4696
Phone_Change                   4696
Credit_Bureau                  4696
Default                        4696
dtype: int64
```

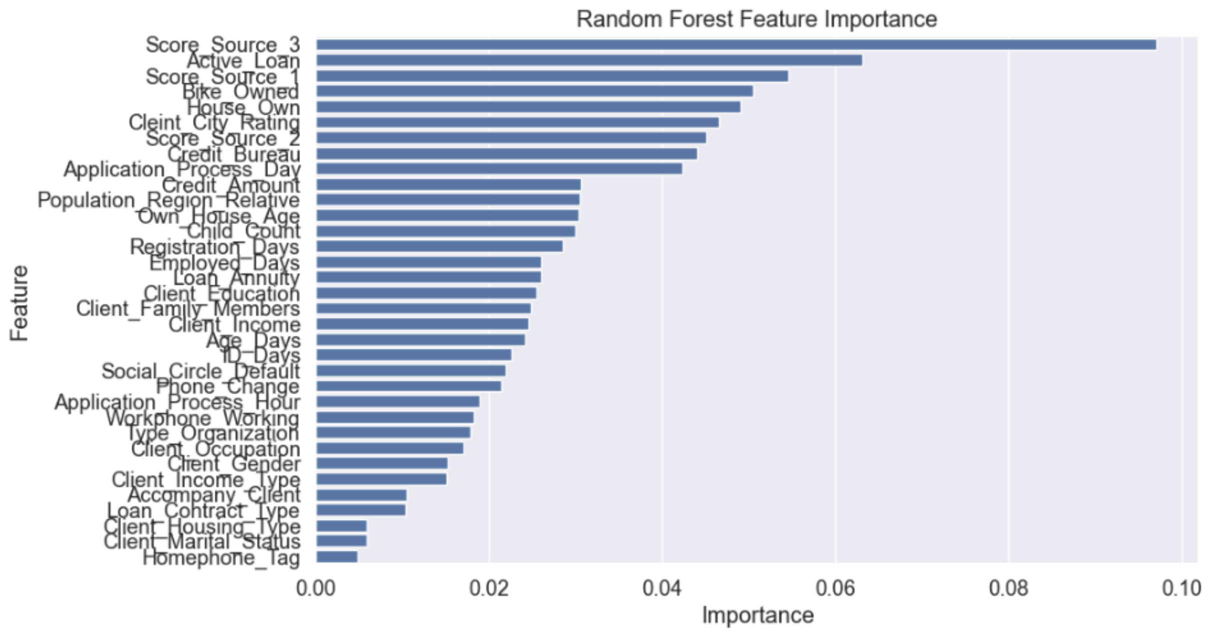
```
[26]: #Splitting the Data
      from sklearn.model_selection import train_test_split

      # Split the balanced dataset into features and target variable
      X_balanced = df_balanced.drop('Default', axis=1)  # Features
      y_balanced = df_balanced['Default']               # Target variable

      # Split into training and testing sets (e.g., 80% train, 20% test)
      X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced)
```

```
[27]: #Feature Scaling
      from sklearn.preprocessing import StandardScaler

      # Initialize StandardScaler
      scaler = StandardScaler()

      # Fit and transform the training data
      X_train_scaled = scaler.fit_transform(X_train)

      # Transform the testing data
      X_test_scaled = scaler.transform(X_test)
```

1. Random Forest Classifier

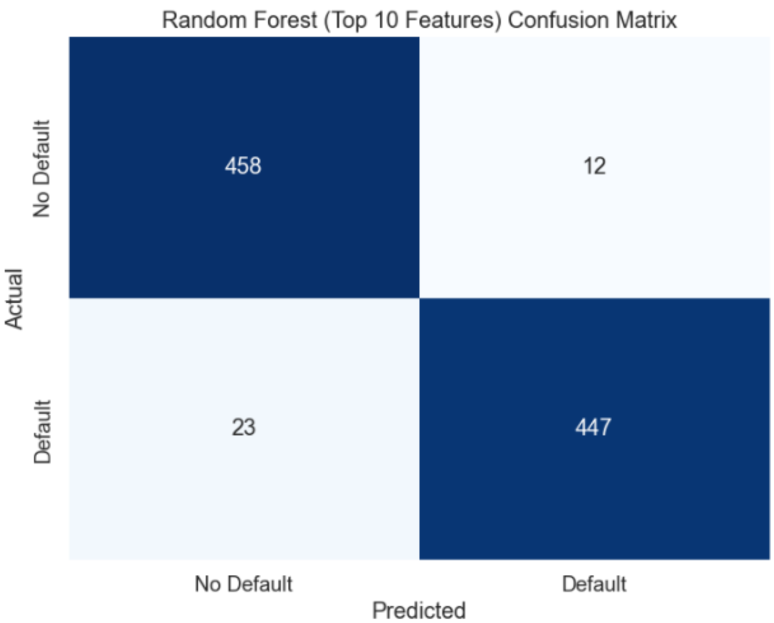
```
[28]: # RandomForest Model

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import joblib

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # Use the original training set here
X_test_scaled = scaler.transform(X_test)         # Use the original test set here

# Convert to DataFrame for easy indexing
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Initialize the RandomForest classifier
rf_model = RandomForestClassifier(random_state=42)

# Train the model on the training data
rf_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_rf = rf_model.predict(X_test_scaled)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest Accuracy: {accuracy_rf:.2f}')

# Create confusion matrix
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print('Random Forest Confusion Matrix:\n', conf_matrix_rf)

# Classification report
class_report_rf = classification_report(y_test, y_pred_rf)
print('Random Forest Classification Report:\n', class_report_rf)

# Get feature importances
importances = rf_model.feature_importances_

# Create a dataframe for feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': importances
})

# Sort by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Plot the feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Random Forest Feature Importance')
plt.show()
```

```
Random Forest Accuracy: 0.98
Random Forest Confusion Matrix:
 [[464   6]
 [ 13 457]]
Random Forest Classification Report:
               precision    recall  f1-score   support

        -1.0       0.97      0.99      0.98       470
         1.0       0.99      0.97      0.98       470

    accuracy                           0.98       940
   macro avg       0.98      0.98      0.98       940
weighted avg       0.98      0.98      0.98       940
```



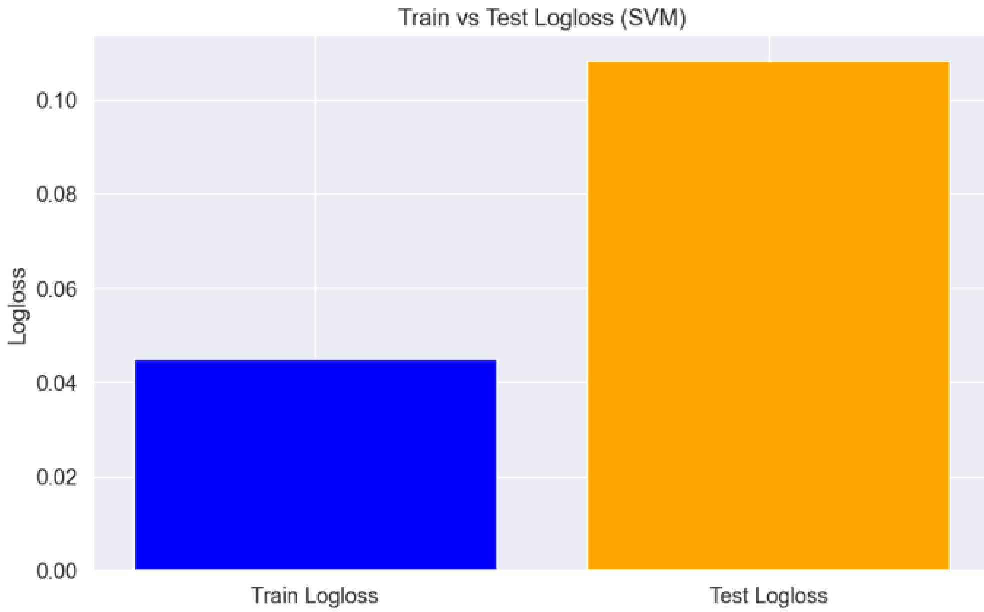

```
[29]: # Identify top features
n_top_features = 10
top_features = feature_importance_df.head(n_top_features)['Feature'].tolist()
print("Top Features:", top_features)

# Filter the training and testing sets for the top features
X_train_top10 = X_train_scaled[top_features]
X_test_top10 = X_test_scaled[top_features]

# Initialize a new RandomForestClassifier
rf_model_top10 = RandomForestClassifier(random_state=42)

# Train the model on the filtered training data
rf_model_top10.fit(X_train_top10, y_train)

# Make predictions on the filtered test data
y_pred_rf_top10 = rf_model_top10.predict(X_test_top10)

# Evaluate the new model
accuracy_rf_top10 = accuracy_score(y_test, y_pred_rf_top10)
print(f'Random Forest (Top 10 Features) Accuracy: {accuracy_rf_top10:.2f}')

# Create a new confusion matrix
conf_matrix_rf_top10 = confusion_matrix(y_test, y_pred_rf_top10)
print('Random Forest (Top 10 Features) Confusion Matrix:\n', conf_matrix_rf_top10)

# Classification report
class_report_rf_top10 = classification_report(y_test, y_pred_rf_top10)
print('Random Forest (Top 10 Features) Classification Report:\n', class_report_rf_top10)

# Plot confusion matrix for the new model
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf_top10, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['No Default', 'Default'], yticklabels=['No Default', 'Default'])
plt.title('Random Forest (Top 10 Features) Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Top Features: ['Score_Source_3', 'Active_Loan', 'Score_Source_1', 'Bike_Owned', 'House_Own', 'Cleint_City_Rating', 'Score_Source_2', 'Credit_Bureau', 'Application_Process_Day', 'Credit_Amount']
Random Forest (Top 10 Features) Accuracy: 0.96
Random Forest (Top 10 Features) Confusion Matrix:
 [[458  12]
 [ 23 447]]
Random Forest (Top 10 Features) Classification Report:
              precision    recall  f1-score   support

        -1.0       0.95      0.97      0.96       470
         1.0       0.97      0.95      0.96       470

    accuracy                           0.96       940
   macro avg       0.96      0.96      0.96       940
weighted avg       0.96      0.96      0.96       940
```

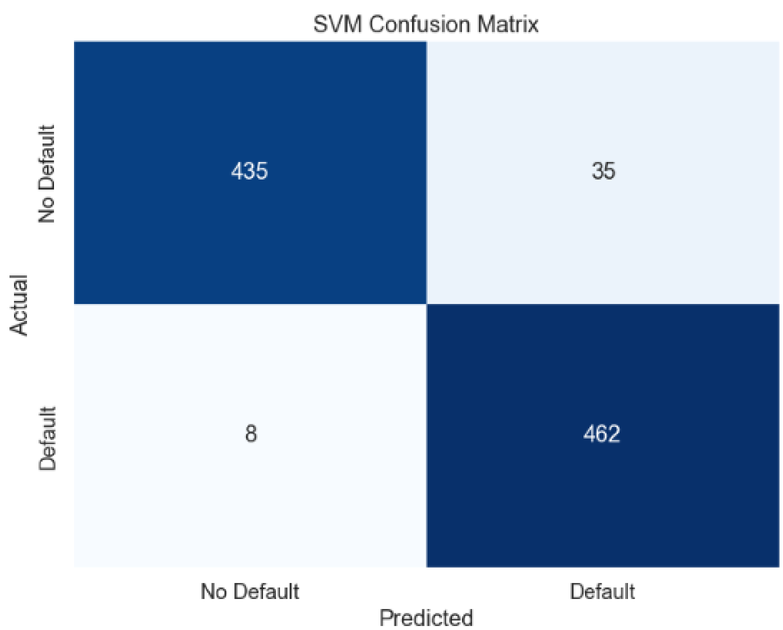


2. Support Vector classifier(SVM)

```
[33]: #SVM Model

# Import necessary libraries
from sklearn.svm import SVC

# Initialize the Support Vector Classifier (SVC) with probability=True for log loss
svc_model = SVC(probability=True, random_state=42)

# Train the model on the training data
svc_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_svc = svc_model.predict(X_test_scaled)

# Evaluate the model
accuracy_svc = accuracy_score(y_test, y_pred_svc)
print(f'SVM Accuracy: {accuracy_svc:.2f}')

# Confusion matrix
conf_matrix_svc = confusion_matrix(y_test, y_pred_svc)
print('SVM Confusion Matrix:\n', conf_matrix_svc)

# Classification report
class_report_svc = classification_report(y_test, y_pred_svc)
print('SVM Classification Report:\n', class_report_svc)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_svc, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['No Default', 'Default'], yticklabels=['No Default', 'Default'])
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# SVM Log Loss per iteration
y_train_pred_prob_svc = svc_model.predict_proba(X_train_scaled)
y_test_pred_prob_svc = svc_model.predict_proba(X_test_scaled)

# Calculate Log Loss
log_loss_train_svc = log_loss(y_train, y_train_pred_prob_svc)
log_loss_test_svc = log_loss(y_test, y_test_pred_prob_svc)

# Plot Train vs Test Log Loss
plt.figure(figsize=(10, 6))
plt.bar(['Train Logloss', 'Test Logloss'], [log_loss_train_svc, log_loss_test_svc], color=['blue', 'orange'])
plt.title('Train vs Test Logloss (SVM)')
plt.ylabel('Logloss')
plt.show()
```



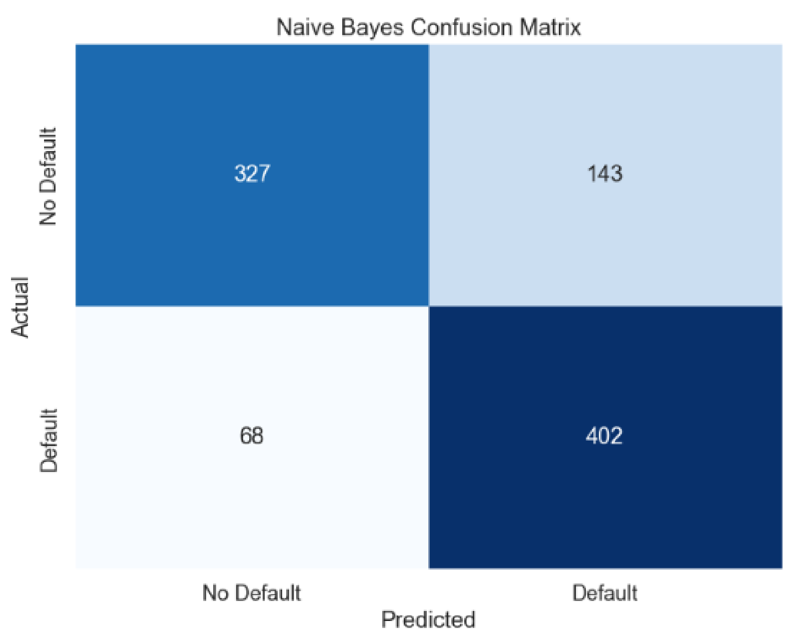

```
SVM Accuracy: 0.95
SVM Confusion Matrix:
 [[435  35]
 [  8 462]]
SVM Classification Report:
               precision    recall  f1-score   support

         0.0       0.98      0.93      0.95       470
         1.0       0.93      0.98      0.96       470

    accuracy                           0.95       940
   macro avg       0.96      0.95      0.95       940
weighted avg       0.96      0.95      0.95       940
```



3. Naïve Bayes Classifier

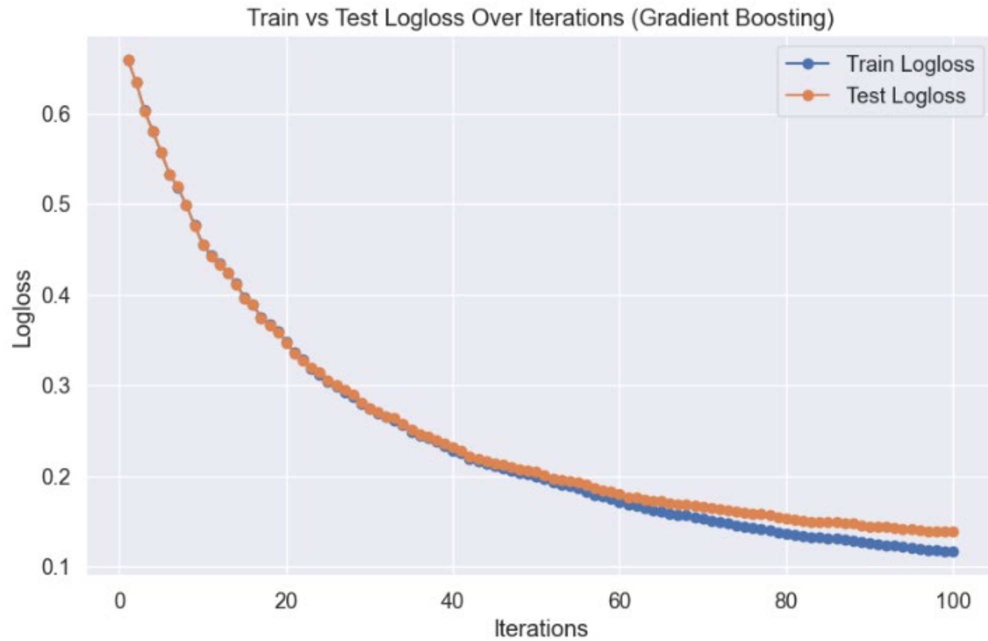
```
[36]: #Naive Bayes Classifier

from sklearn.naive_bayes import GaussianNB

# Initialize the Naive Bayes classifier
nb_model = GaussianNB()

# Train the model
nb_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_nb = nb_model.predict(X_test_scaled)

# Evaluate the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f'Naive Bayes Accuracy: {accuracy_nb:.2f}')

# Confusion matrix
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)
print('Naive Bayes Confusion Matrix:\n', conf_matrix_nb)

# Classification report
class_report_nb = classification_report(y_test, y_pred_nb)
print('Naive Bayes Classification Report:\n', class_report_nb)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_nb, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['No Default', 'Default'], yticklabels=['No Default', 'Default'])
plt.title('Naive Bayes Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Naive Bayes Accuracy: 0.78
Naive Bayes Confusion Matrix:
 [[327 143]
 [ 68 402]]
Naive Bayes Classification Report:
               precision    recall  f1-score   support

         0.0       0.83      0.70      0.76       470
         1.0       0.74      0.86      0.79       470

    accuracy                           0.78       940
   macro avg       0.78      0.78      0.77       940
weighted avg       0.78      0.78      0.77       940
```

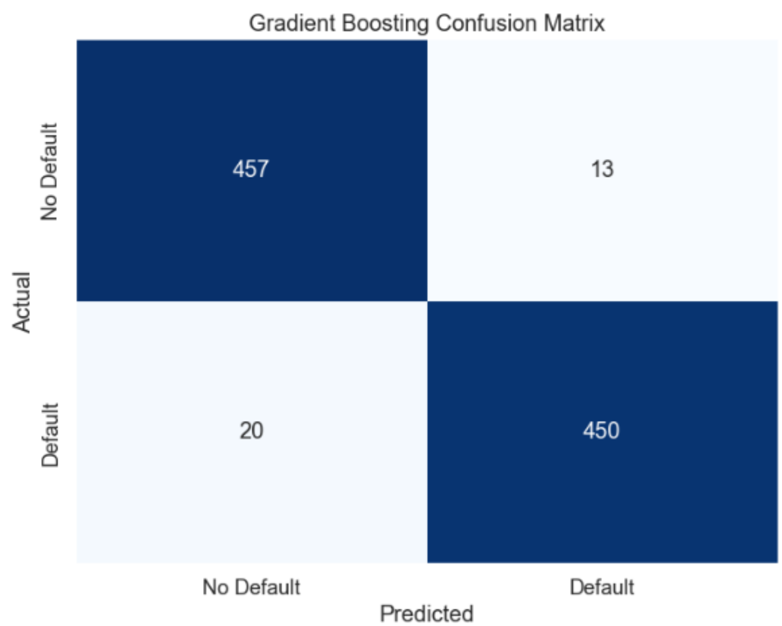


4. Gradient Boosting Classifier

```
[32]: #Gradient Boosting Classifier

# Import necessary libraries
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import log_loss
import seaborn as sns

# Initialize the Gradient Boosting classifier
gbc_model = GradientBoostingClassifier(random_state=42)

# Train the model on the training data
gbc_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_gbc = gbc_model.predict(X_test_scaled)

# Evaluate the model
accuracy_gbc = accuracy_score(y_test, y_pred_gbc)
print(f'Gradient Boosting Accuracy: {accuracy_gbc:.2f}')

# Confusion matrix
conf_matrix_gbc = confusion_matrix(y_test, y_pred_gbc)
print('Gradient Boosting Confusion Matrix:\n', conf_matrix_gbc)

# Classification report
class_report_gbc = classification_report(y_test, y_pred_gbc)
print('Gradient Boosting Classification Report:\n', class_report_gbc)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_gbc, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['No Default', 'Default'], yticklabels=['No Default', 'Default'])
plt.title('Gradient Boosting Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Gradient Boosting Log Loss per iteration (staged prediction)
log_loss_train = []
log_loss_test = []

# Get the predicted probabilities and compute log loss at each iteration
for y_train_pred_prob, y_test_pred_prob in zip(gbc_model.staged_predict_proba(X_train_scaled),
                                               gbc_model.staged_predict_proba(X_test_scaled)):
    log_loss_train.append(log_loss(y_train, y_train_pred_prob))
    log_loss_test.append(log_loss(y_test, y_test_pred_prob))

# Plot Train vs Test Log Loss Over Iterations
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(log_loss_train) + 1), log_loss_train, label='Train Logloss', marker='o')
plt.plot(range(1, len(log_loss_test) + 1), log_loss_test, label='Test Logloss', marker='o')
plt.title('Train vs Test Logloss Over Iterations (Gradient Boosting)')
plt.xlabel('Iterations')
plt.ylabel('Logloss')
plt.legend()
plt.show()
```



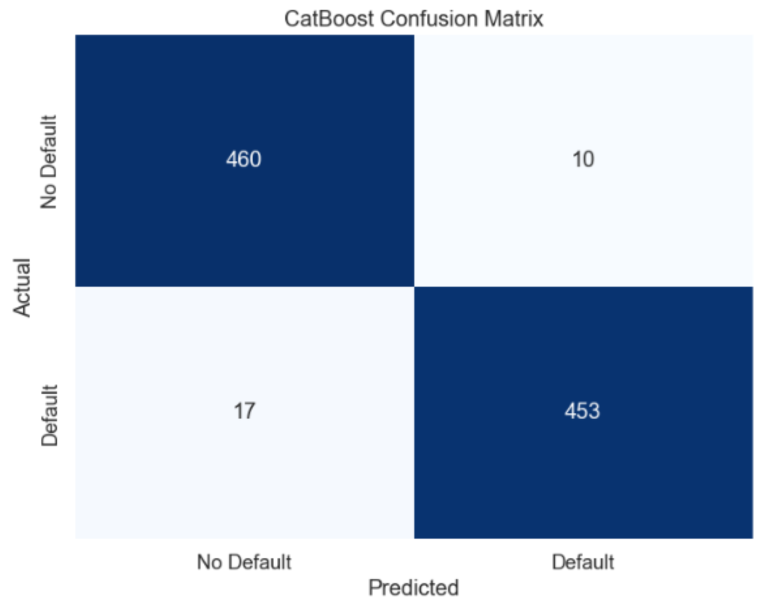

```
Gradient Boosting Accuracy: 0.96
Gradient Boosting Confusion Matrix:
 [[457  13]
 [ 20 450]]
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.97       470
         1.0       0.97      0.96      0.96       470

    accuracy                           0.96       940
   macro avg       0.96      0.96      0.96       940
weighted avg       0.96      0.96      0.96       940
```

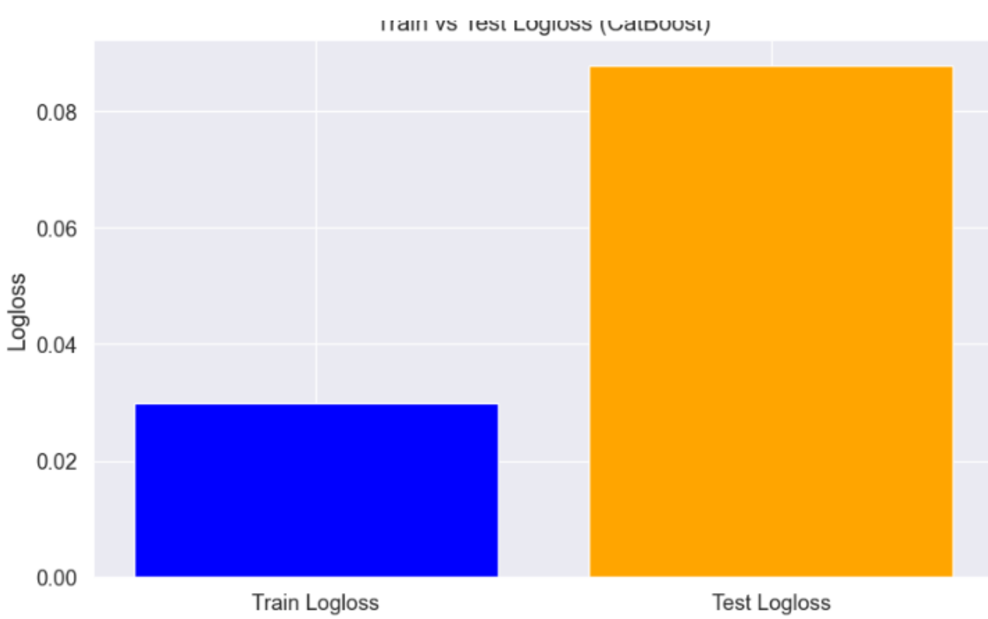


5. Cat Boost Classifier

```
[35]: #CatBoost Classifier

# Import necessary libraries
from catboost import CatBoostClassifier

# Initialize the CatBoost Classifier
catboost_model = CatBoostClassifier(random_state=42, verbose=0)

# Train the model on the training data
catboost_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_catboost = catboost_model.predict(X_test_scaled)
y_pred_prob_catboost = catboost_model.predict_proba(X_test_scaled)

# Evaluate the model
accuracy_catboost = accuracy_score(y_test, y_pred_catboost)
print(f'CatBoost Accuracy: {accuracy_catboost:.2f}')

# Confusion matrix
conf_matrix_catboost = confusion_matrix(y_test, y_pred_catboost)
print('CatBoost Confusion Matrix:\n', conf_matrix_catboost)

# Classification report
class_report_catboost = classification_report(y_test, y_pred_catboost)
print('CatBoost Classification Report:\n', class_report_catboost)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_catboost, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['No Default', 'Default'], yticklabels=['No Default', 'Default'])
plt.title('CatBoost Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Calculate Log Loss
log_loss_train_catboost = log_loss(y_train, catboost_model.predict_proba(X_train_scaled))
log_loss_test_catboost = log_loss(y_test, catboost_model.predict_proba(X_test_scaled))

# Plot Train vs Test Log Loss
plt.figure(figsize=(10, 6))
plt.bar(['Train Logloss', 'Test Logloss'], [log_loss_train_catboost, log_loss_test_catboost], color=['blue', 'orange'])
plt.title('Train vs Test Logloss (CatBoost)')
plt.ylabel('Logloss')
plt.show()
```

```
CatBoost Accuracy: 0.97
CatBoost Confusion Matrix:
 [[460  10]
 [ 17 453]]
CatBoost Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.98      0.97       470
         1.0       0.98      0.96      0.97       470

    accuracy                           0.97       940
   macro avg       0.97      0.97      0.97       940
weighted avg       0.97      0.97      0.97       940
```

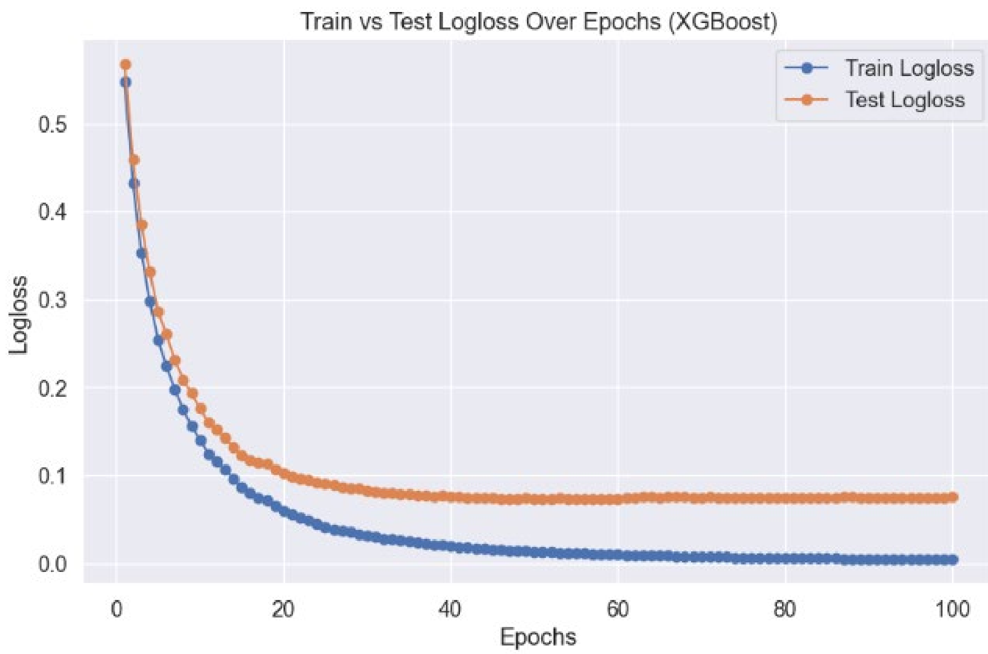


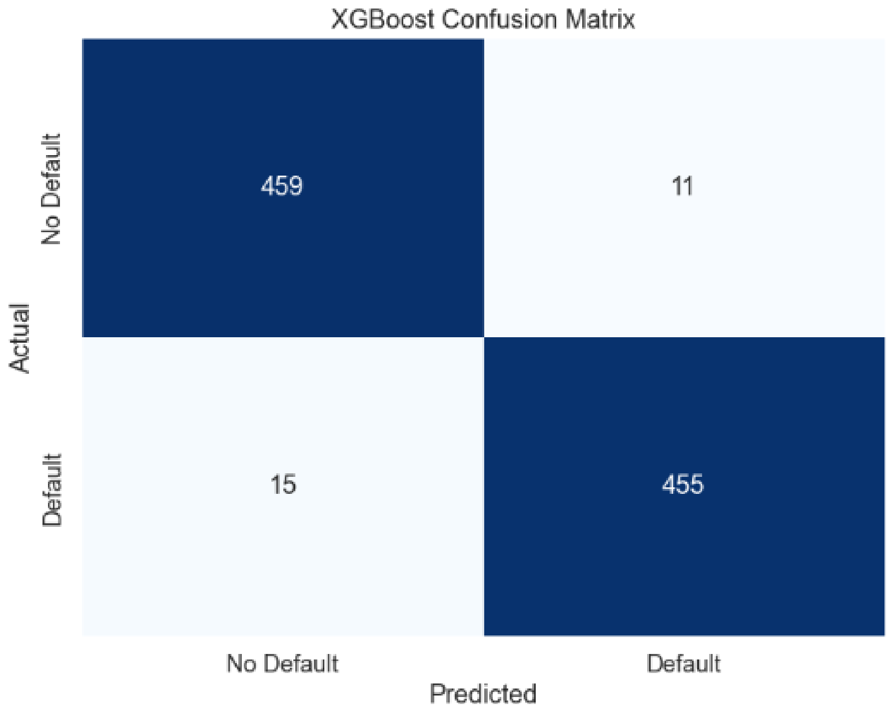


6. XGBoost Classifier

```
[31]: #XGBoost Classifier

# Import necessary libraries
import xgboost as xgb
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize the XGBoost classifier with logging for evaluation metrics
xgb_model = xgb.XGBClassifier(random_state=42, eval_metric="logloss")

# Check unique values in y_train
print("Unique values in y_train before transformation:", set(y_train))

# Transform y_train and y_test if necessary (from -1, 1 to 0, 1)
y_train = (y_train + 1) / 2  # Transforms -1 to 0 and 1 to 1
y_test = (y_test + 1) / 2    # Same transformation for y_test

# Check unique values after transformation
print("Transformed unique values in y_train:", set(y_train))

# Create a watchlist to monitor training and testing performance
eval_set = [(X_train_scaled, y_train), (X_test_scaled, y_test)]

# Train the model on the training data and track logloss
xgb_model.fit(X_train_scaled, y_train, eval_set=eval_set, verbose=False)

# Extract the evaluation results
results = xgb_model.evals_result()

# Make predictions on the test data
y_pred_xgb = xgb_model.predict(X_test_scaled)

# Evaluate the model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f'XGBoost Accuracy: {accuracy_xgb:.2f}')

# Confusion matrix
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
print('XGBoost Confusion Matrix:\n', conf_matrix_xgb)

# Classification report
class_report_xgb = classification_report(y_test, y_pred_xgb)
print('XGBoost Classification Report:\n', class_report_xgb)

# Plot Train vs Test Logloss Over Epochs
epochs = range(1, len(results['validation_0']['logloss']) + 1)

plt.figure(figsize=(10, 6))
plt.plot(epochs, results['validation_0']['logloss'], label='Train Logloss', marker='o')
plt.plot(epochs, results['validation_1']['logloss'], label='Test Logloss', marker='o')
plt.title('Train vs Test Logloss Over Epochs (XGBoost)')
plt.xlabel('Epochs')
plt.ylabel('Logloss')
plt.legend()
plt.show()

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_xgb, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['No Default', 'Default'], yticklabels=['No Default', 'Default'])
plt.title('XGBoost Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Unique values in y_train before transformation: {1.0, -1.0}
Transformed unique values in y_train: {0.0, 1.0}
XGBoost Accuracy: 0.97
XGBoost Confusion Matrix:
 [[459  11]
 [ 15 455]]
XGBoost Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.98      0.97       470
         1.0       0.98      0.97      0.97       470

    accuracy                           0.97       940
   macro avg       0.97      0.97      0.97       940
weighted avg       0.97      0.97      0.97       940
```

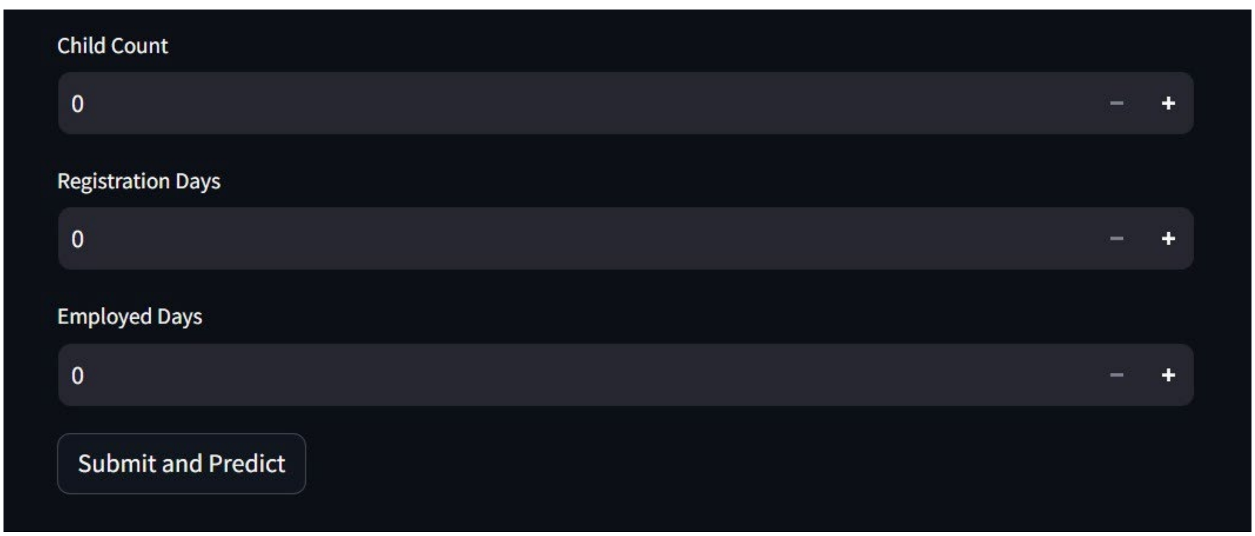



XGBoost Confusion Matrix
No Default
Default
Actual
459
11
15
455
No Default
Default
Predicted

User Interface

The interface is created using Streamlit.


Score Source 2
0.00
Credit Bureau
0
Application Process Day
0
Credit Amount
0.00
Population Region Relative
0
Own House Age
0
Child Count
0


Child Count
0
Registration Days
0
Employed Days
0
Submit and Predict

Benefits of the Proposed Solution

Building a predictive model for loan approval offers a range of benefits for financial institutions and their customers. Here are some key advantages of such a solution:

1.Improved Decision-Making: The predictive model provides data-driven insights to accurately assess the repayment ability of loan applicants. By automating the credit evaluation process, the system helps loan officers make more informed decisions, reducing the risk of defaults.

2.Reduced Loan Default Rate: By identifying high-risk applicants, the model helps minimize the number of loans given to clients who are likely to default. This reduces financial losses and contributes to healthier loan portfolios for vehicle loan companies.

3.Enhanced Efficiency: Traditional credit assessments are often time-consuming and resource-intensive. The proposed solution significantly reduces the time needed to evaluate loan applicants, allowing financial institutions to handle a larger volume of applications more efficiently.

4.Consistency and Reduced Bias: Machine learning models are not influenced by subjective factors like emotions or biases, which can affect human decision-making. This leads to a more consistent and fair credit evaluation process, ensuring that all applicants are assessed based on objective criteria.

5.Cost Savings: Automating the credit risk assessment process can lead to significant cost reductions in loan processing and reduce the overhead associated with manual evaluations. This helps companies allocate resources more effectively.

6.Improved Customer Experience: Faster and more accurate decisions lead to quicker responses to loan applicants, enhancing the overall customer experience. A smoother loan approval process can also boost customer satisfaction and attract more clients.

Conclusion

In an increasingly competitive financial landscape, vehicle loan companies should adapt to changing demands by embracing data-driven decision-making. The suggested prediction model provides an innovative method to evaluating each loan applicant's risk and provides an accurate assessment of the possibility of repayment.. By using client data effectively, the system not only improves decision accuracy but also minimizes defaults, ensuring a more profitable loan portfolio for financial institutions. The automation of credit assessments addresses the limitations of traditional methods by enhancing both efficiency and consistency in decision-making.

Scalability is a key strength of this solution, enabling companies to meet growing market demand without compromising quality or increasing operational costs. As the vehicle loan sector expands, the ability to handle more applications efficiently while maintaining accuracy becomes crucial. By preventing high-risk loans, financial institutions can significantly reduce the rate of non-performing loans, thus minimizing financial losses and fostering a sustainable business model. Additionally, automated assessment tools allow for faster processing times, enhancing customer experience by delivering prompt and transparent decisions.

Another vital aspect of the proposed solution is its role in customer acquisition and retention. By providing rapid and consistent loan decisions, vehicle loan companies can position themselves as customer-centric, ultimately leading to higher satisfaction rates and increased loyalty. Furthermore, the model's alignment with responsible lending practices and regulatory compliance ensures that companies maintain financial stability while meeting industry standards. Leveraging historical data also enables institutions to make more informed strategic decisions and extract valuable insights from existing datasets.

Overall, the proposed predictive system provides a significant improvement in the credit assessment process, enhancing decision-making capabilities, operational efficiency, and customer experience. By adopting this advanced, data-driven approach, vehicle loan companies can better manage risk, optimize their operations, and create a resilient business model capable of adapting to future growth and challenges.

Project Team, Roles, and Responsibilities

Ranasinghe R.Y.G(IT22253880)	• Model building • Model Optimization • Deploying the model • Design UI
Gamage M.P.L (IT22578082)	• Feature Selection • Feature Engineering • Exploratory Analysis • Design UI
Thiyanima H.E.S(IT22271600)	• Data visualization • Data preprocessing • Complete project documentation • Design UI
Dilshan H.M.T.W(IT22562456)	• Deploying the model • Design UI • Feature Engineering • Addresses fairness concerns in the data and models
Handapangoda C.N(IT22586070)	• Data visualization • Model Evaluation • Complete project documentation • Design UI