Group 3: - ARFAAT, ARSHEEN, JOEHAN

Loan Approval Prediction Web App

Introduction:

Loans are the major requirement of the modern world. By this only, Banks get a major part of the total profit. It is beneficial for students to manage their education and living expenses, and for people to buy any kind of luxury like houses, cars, etc.

But when it comes to deciding whether the applicant's profile is relevant to be granted with loan or not. Banks have to look after many aspects.

So, here we will be using Machine Learning with Python to ease their work and predict whether the candidate's profile is relevant or not using key features like Marital Status, Education, Applicant Income, Credit History, etc.

This Streamlit-based web application predicts loan approval status based on user inputs. The model used is a RandomForestClassifier, trained on a dataset containing various applicant details.

Features

- User-friendly interface for inputting applicant details.
- Encoding and processing of categorical variables.
- Handling of missing values.
- Display of loan approval status based on model predictions.
- Visualization of data overview and encoding mappings.

Installation

1. Clone the repository or download the project files.
2. Ensure you have Python installed (version 3.6 or higher recommended).
3. Install the required packages using pip:
   ```
   PIP INSTALL STREAMLIT PANDAS NUMPY MATPLOTLIB SEABORN SCIKIT-LEARN
   ```
4. Download the dataset LoanApprovalPrediction.csv and place it in the project directory.

Usage

- Run the Streamlit app:
  ```
  STREAMLIT RUN APP.PY
  ```
- Open your browser and navigate to the local Streamlit server (usually http://localhost:8501).
- Input the required applicant details in the provided fields.
- Click the "Predict" button to see the loan approval status.

Programming Language and Libraries Used:

1. **Programming Language**

- **Python:** Python is a versatile, high-level programming language known for its readability and efficiency. Python's syntax is clear and intuitive, making it accessible for beginners while powerful enough for experts. It supports various programming paradigms, including procedural, object-oriented, and functional programming. Python is extensively used in data science, web development, automation, and machine learning due to its rich ecosystem of libraries and frameworks.

2. **Libraries**

- **Pandas:** Pandas is a powerful library for data manipulation and analysis, providing data structures like DataFrame. It is designed for working with structured data efficiently and intuitively. Pandas supports operations for data cleaning, merging, reshaping, and aggregation, which are fundamental in data preprocessing and analysis. It seamlessly integrates with other libraries in the Python ecosystem, making it a cornerstone for data science projects**.**

- **NumPy:** NumPy, short for Numerical Python, is a fundamental package for scientific computing in Python. It supports large, multi-dimensional arrays and matrices along with a collection of mathematical functions to operate on these arrays. NumPy serves as the base for other scientific libraries like Pandas and Scikit-learn, providing performance optimizations for numerical computations.

- **Matplotlib:** Matplotlib is a plotting library for creating static, interactive, and animated visualizations in Python. It provides an object-oriented API for embedding plots into applications. Matplotlib is highly customizable, allowing users to create a wide variety of plots and charts to visualize data effectively. It is often used in conjunction with libraries like NumPy and Pandas for data visualization.

- **Scikit-learn:** Scikit-learn is a machine learning library that provides simple and efficient tools for data mining and data analysis. It is built on NumPy, SciPy, and Matplotlib. Scikit-learn offers a wide range of supervised and unsupervised learning algorithms, along with tools for model selection, evaluation, and preprocessing. Its consistent and user-friendly API makes it a popular choice for both beginners and experts in machine learning.

- **Imbalanced-learn:** Imbalanced-learn is a library offering tools to deal with imbalanced data sets in machine learning. It provides various resampling techniques, such as over-sampling, under-sampling, and generating synthetic samples (like SMOTE). These techniques help in balancing the distribution of classes, improving the model's ability to learn from minority classes, and enhancing overall prediction performance. Imbalanced-learn integrates well with Scikit-learn, enabling seamless incorporation of its techniques into machine learning workflows.

Group 3: - ARFAAT, ARSHEEN, JOEHAN

Understanding The Data:

Overview of the Data:

1	Loan	A unique id
2	Gender	Gender of the applicant Male/female
3	Married	Marital Status of the applicant, values will be Yes/ No
4	Dependents	It tells whether the applicant has any dependents or not.
5	Education	It will tell us whether the applicant is Graduated or not.
6	Self_Employed	This defines that the applicant is self-employed i.e. Yes/ No
7	ApplicantIncome	Applicant income
8	CoapplicantIncome	Co-applicant income
9	LoanAmount	Loan amount (in thousands)
10	Loan_Amount_Term	Terms of loan (in months)
11	Credit_History	Credit history of individual's repayment of their debts
12	Property_Area	Area of property i.e. Rural/Urban/Semi-urban
13	Loan_Status	Status of Loan Approved or not i.e. Y- Yes, N-No

Sample of the Dataset:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed
0	LP001002	Male	No	0.0	Graduate	No
1	LP001003	Male	Yes	1.0	Graduate	No
2	LP001005	Male	Yes	0.0	Graduate	Yes
3	LP001006	Male	Yes	0.0	Not Graduate	No
4	LP001008	Male	No	0.0	Graduate	No

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	5849	0.0	NaN	360.0
1	4583	1508.0	128.0	360.0
2	3000	0.0	66.0	360.0
3	2583	2358.0	120.0	360.0
4	6000	0.0	141.0	360.0

	Credit_History	Property_Area	Loan_Status
0	1.0	Urban	Y
1	1.0	Rural	N
2	1.0	Urban	Y
3	1.0	Urban	Y
4	1.0	Urban	Y

As we can see here there are 7 Categorical Variables Namely Loan_ID, Gender, Education, Self_Employed, Credit_History, Property_Area and Loan_Status

As Loan_ID is completely unique and not correlated with any of the other column, So we will drop it using .drop() function.

```
# Dropping Loan_ID column
data.drop(['Loan_ID'], axis=1, inplace=True)
```

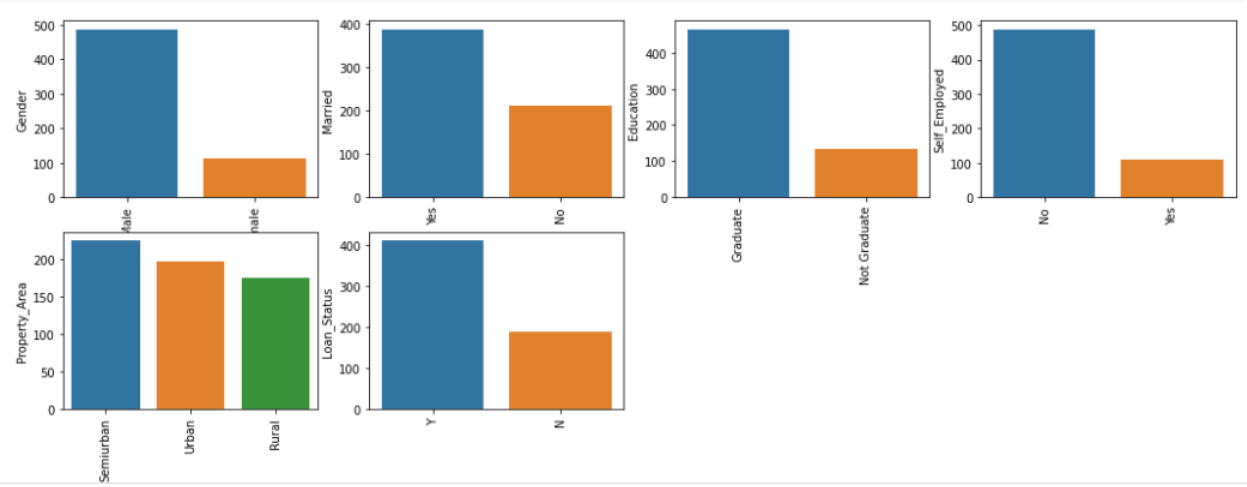
Understanding The Categorical Data:

We can begin by visualizing all the unique values in columns using barplot. This will simply show which value is dominating as per our dataset.

```
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)
plt.figure(figsize=(18,36))
index = 1

for col in object_cols:
  y = data[col].value_counts()
  plt.subplot(11,4,index)
  plt.xticks(rotation=90)
  sns.barplot(x=list(y.index), y=y)
  index +=1
```

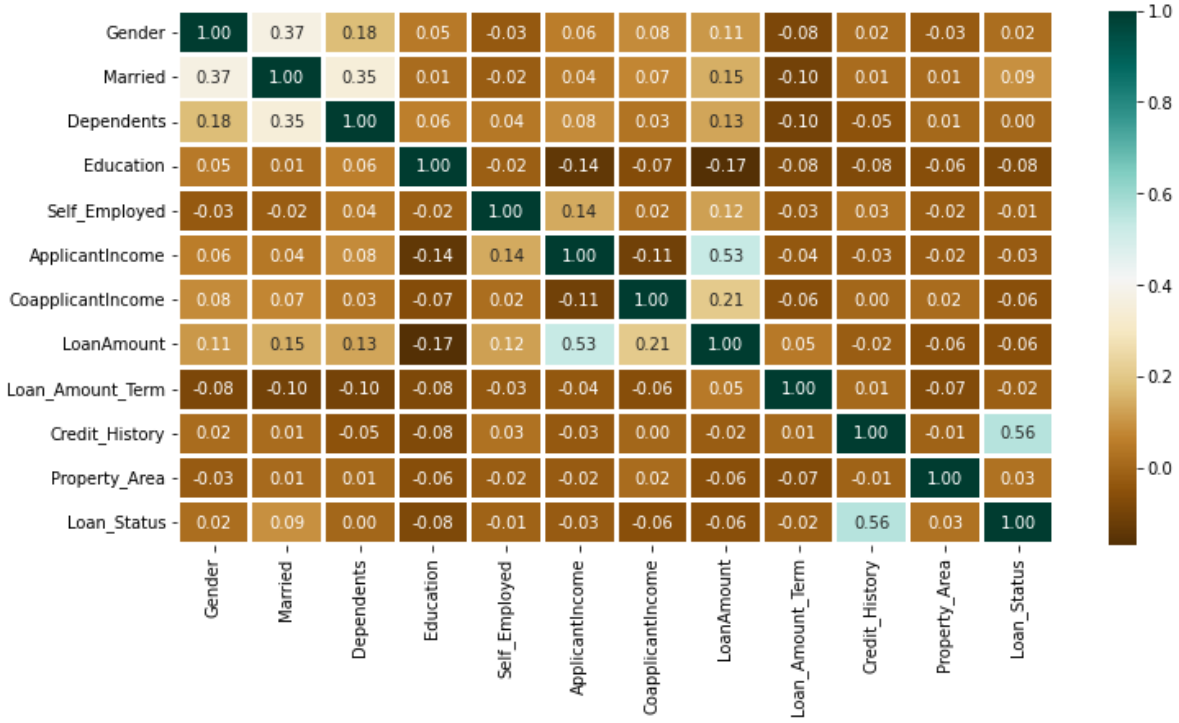
The following Plot is the output of the code given Above:



As all the categorical values are binary so we can use Label Encoder in our program for all such columns and the values will change into int datatype.

Understanding the Correlation Between Variables:

```
plt.figure(figsize=(12,6))
sns.heatmap(data.corr(),cmap='BrBG',fmt='.2f', linewidths=2,annot=True)
```



The above heatmap shows there is high correlation between Loan Amount and Applicant Income. It also shows that Credit_History has a high impact on Loan_Status.

Model Evaluation and Selection:

As this is a classification problem, we will be using these models:

- KNeighborsClassifiers
- RandomForestClassifiers
- Support Vector Classifiers (SVC)
- Logistics Regression

To predict the accuracy, we will use the accuracy score function from scikit-learn library.

The Following is the code to Evaluate the Various Model:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
```

```
from sklearn import metrics

knn = KNeighborsClassifier(n_neighbors=3)
rfc = RandomForestClassifier(n_estimators = 7, criterion = 'entropy',
random_state =7)
svc = SVC()
lc = LogisticRegression()

# making predictions on the training set
for clf in (rfc, knn, svc,lc):
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_train)
    print("Accuracy score of ", clf.__class__.__name__,
"=",100*metrics.accuracy_score(Y_train, Y_pred))
```

Output :

Accuracy score of RandomForestClassifier = 98.04469273743017

Accuracy score of KNeighborsClassifier = 78.49162011173185

Accuracy score of SVC = 68.71508379888269

Accuracy score of LogisticRegression = 80.44692737430168

Prediction on the test set:

```
# making predictions on the testing set
for clf in (rfc, knn, svc,lc):
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_test)
    print("Accuracy score of ", clf.__class__.__name__,"=",
100*metrics.accuracy_score(Y_test, Y_pred))
```

Output :

Accuracy score of RandomForestClassifier = 82.5

Accuracy score of KNeighborsClassifier = 63.74999999999999

Accuracy score of SVC = 69.16666666666667

Accuracy score of LogisticRegression = 80.83333333333333

Conclusion: From this we can Conclude that Random Forest Classifier is the better of the other models giving the best accuracy with an accuracy score of 82% for the testing dataset.

Understanding The Code:

Imports:

```
from imblearn.over_sampling import SMOTE
import streamlit as st
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.impute import SimpleImputer
```

Data Preparation:

- **Reading the Data:** The data is read from a CSV file using Pandas.

```
# Reading the data
data = pd.read_csv("LoanApprovalPrediction.csv")
```

- **Dropping Unnecessary Columns:** The Loan_ID column is dropped as it's not needed for prediction.

```
# Dropping Loan_ID column
data.drop(['Loan_ID'], axis=1, inplace=True)
```

- **Encoding Categorical Columns:** Categorical columns are encoded using LabelEncoder to convert text data into numerical form.

```
# Creating label encoder object
label_encoder = preprocessing.LabelEncoder()

# Finding categorical columns
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)

# Encoding categorical columns and storing mappings
encoding_maps = {}
for col in object_cols:
    data[col] = label_encoder.fit_transform(data[col])
    encoding_maps[col] = dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_)))
```

Model Training

Splitting the Data: The dataset is split into training and testing sets.

```
# Splitting the data into train and test sets
X = data_imputed.drop(['Loan_Status'], axis=1)
Y = data_imputed['Loan_Status']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4,
random_state=1)
```

Imputing Missing Values: Missing values are imputed using SimpleImputer with the mean strategy.

```
# Imputing missing values
imputer = SimpleImputer(strategy='mean')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
```

Resampling: Resamples the training data using SMOTE (Synthetic Minority Over-sampling Technique) to address class imbalance.

```
# Applying SMOTE to the training data
smote = SMOTE(random_state=1)
X_resampled, Y_resampled = smote.fit_resample(X_train, Y_train)
```

Training the Model: A RandomForestClassifier is trained on the training data.

```
# Training the RandomForest model with the resampled data
rfc = RandomForestClassifier(n_estimators=7, criterion='entropy', random_state=7)
rfc.fit(X_resampled, Y_resampled)
```

Testing and Evaluating the Model:

```
# Predicting on the test set and evaluating
Y_pred = rfc.predict(X_test)
accuracy = 100 * metrics.accuracy_score(Y_test, Y_pred)
print("Accuracy: ", accuracy)
```

Streamlit App

Data Overview: Displays the first few rows of the dataset.

```
# Streamlit app
st.title("Loan Approval Prediction")

st.header("Data Overview")
st.write(X_resampled.head(15))
```

Encoding Mappings: Shows the mappings used for encoding categorical variables.

```
st.header("Encoding Mappings")
for col, encoding_map in encoding_maps.items():
    st.write(f'**Encoding for {col}:**')
    for original, encoded in encoding_map.items():
        st.write(f'{original}: {encoded}')
    st.write('')
```

Loan Approval Prediction: Takes user input, processes it, and displays the loan approval status.

```
st.header("Loan Approval Prediction")
gender = st.selectbox("Gender", options=["Male", "Female"])
married = st.selectbox("Married", options=["Yes", "No"])
dependents = st.selectbox("Dependents", options=["0", "1", "2", "3+"])
education = st.selectbox("Education", options=["Graduate", "Not Graduate"])
self_employed = st.selectbox("Self Employed", options=["Yes", "No"])
applicant_income = st.number_input("Applicant Income", value=0)
coapplicant_income = st.number_input("Coapplicant Income", value=0)
loan_amount = st.number_input("Loan Amount", value=0)
loan_amount_term = st.number_input("Loan Amount Term", value=360)
credit_history = st.selectbox("Credit History", options=[0.0, 1.0])
property_area = st.selectbox("Property Area", options=["Urban", "Semiurban",
"Rural"])

# Encode and predict
# ...

if st.button("Predict"):
    prediction = rfc.predict(user_input)[0]
    status = "Congratulations! Your loan is Approved. 🎉" if prediction == 1
else "Sorry, your loan is Not Approved. 😔"
    st.write(f"Loan Status: {status}")
```

Complete Code: Finally the complete Program

```
from imblearn.over_sampling import SMOTE
import streamlit as st
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.impute import SimpleImputer

# Reading the data
data = pd.read_csv("LoanApprovalPrediction.csv")

# Dropping Loan_ID column
data.drop(['Loan_ID'], axis=1, inplace=True)

# Creating label encoder object
label_encoder = preprocessing.LabelEncoder()

# Finding categorical columns
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)

# Encoding categorical columns and storing mappings
encoding_maps = {}
for col in object_cols:
    data[col] = label_encoder.fit_transform(data[col])
    encoding_maps[col] = dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_)))

# Imputing missing values
imputer = SimpleImputer(strategy='mean')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

# Splitting the data into train and test sets
X = data_imputed.drop(['Loan_Status'], axis=1)
Y = data_imputed['Loan_Status']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4,
random_state=1)

# Applying SMOTE to the training data
smote = SMOTE(random_state=1)
X_resampled, Y_resampled = smote.fit_resample(X_train, Y_train)
```

Group 3: - ARFAAT, ARSHEEN, JOEHAN

```
# Training the RandomForest model with the resampled data
rfc = RandomForestClassifier(n_estimators=7, criterion='entropy', random_state=7)
rfc.fit(X_resampled, Y_resampled)

# Predicting on the test set and evaluating
Y_pred = rfc.predict(X_test)
accuracy = 100 * metrics.accuracy_score(Y_test, Y_pred)
print("Accuracy: ", accuracy)

# Streamlit app
st.title("Loan Approval Prediction")

st.header("Data Overview")
st.write(X_resampled.head(15))

st.header("Encoding Mappings")
for col, encoding_map in encoding_maps.items():
    st.write(f'**Encoding for {col}:**')
    for original, encoded in encoding_map.items():
        st.write(f'{original}: {encoded}')
    st.write('')

st.header("Loan Approval Prediction")
gender = st.selectbox("Gender", options=["Male", "Female"])
married = st.selectbox("Married", options=["Yes", "No"])
dependents = st.selectbox("Dependents", options=["0", "1", "2", "3+"])
education = st.selectbox("Education", options=["Graduate", "Not Graduate"])
self_employed = st.selectbox("Self Employed", options=["Yes", "No"])
applicant_income = st.number_input("Applicant Income", value=0)
coapplicant_income = st.number_input("Coapplicant Income", value=0)
loan_amount = st.number_input("Loan Amount", value=0)
loan_amount_term = st.number_input("Loan Amount Term", value=360)
credit_history = st.selectbox("Credit History", options=[0.0, 1.0])
property_area = st.selectbox("Property Area", options=["Urban", "Semiurban",
"Rural"])

# Create input DataFrame for prediction
user_input = pd.DataFrame({
    'Gender': [gender],
    'Married': [married],
    'Dependents': [dependents],
    'Education': [education],
    'Self_Employed': [self_employed],
    'ApplicantIncome': [applicant_income],
```

```
    'CoapplicantIncome': [coapplicant_income],
    'LoanAmount': [loan_amount],
    'Loan_Amount_Term': [loan_amount_term],
    'Credit_History': [credit_history],
    'Property_Area': [property_area]
})

# Encode user input using the same mappings
for col in user_input.columns:
    if col in encoding_maps:
        user_input[col] = user_input[col].map(encoding_maps[col])

# Handle any potential NaN values after mapping
user_input.fillna(-1, inplace=True)

# Predict
if st.button("Predict"):
    prediction = rfc.predict(user_input)[0]
    status = "Congratulations! Your loan is Approved. 🎉" if prediction == 1
else "Sorry, your loan is Not Approved. 😔"
    st.write(f"Loan Status: {status}")
```

Group 3: - ARFAAT, ARSHEEN, JOEHAN

Results

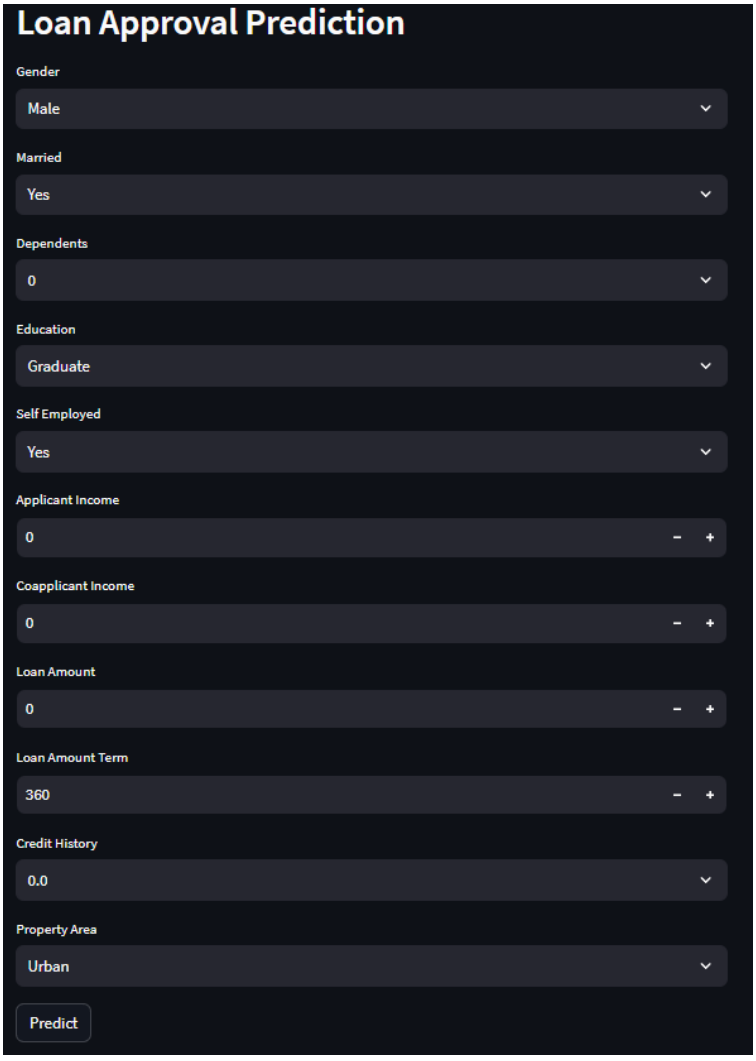
After implementing the RandomForestClassifier and addressing class imbalance using SMOTE, the model achieved an accuracy score of approximately 82.61% on the test set. This score indicates that the model is fairly accurate in predicting loan approval status based on the given applicant details.

Key highlights include:

Model Accuracy: The accuracy score of the RandomForest model after resampling is 82.61%.

Bias Handling: Class imbalance in the dataset was mitigated using SMOTE, which generated synthetic samples for the minority class to balance the data.

User Interaction: The Streamlit app provides an interactive interface for users to input applicant details and receive real-time predictions on loan approval status.



For a given input, the model provides a score or a classification:

Score: A probability that the loan will be approved (e.g., 0.85 means an 85% chance of approval).

Classification: The model may return "Approved" or "Rejected" based on a set threshold (e.g., loans with a probability greater than 0.7 might be classified as "Approved").

Reference

1. https://www.geeksforgeeks.org/loan-approval-prediction-using-machine-learning/

2. https://www.geeksforgeeks.org/loan-approval-prediction-using-machine-learning/?ref=lbp

3. https://www.geeksforgeeks.org/loan-eligibility-prediction-using-machine-learning-models-in-python/?ref=lbp