LOAN-APPROVAL PREDICTION PROJECT IN MACHINE LEARNING

Parth Mishra

OBJECTIVE

The Loan Approval Prediction project aims to automate the loan approval process using machine learning algorithms. It analyzes various applicant factors such as income, credit score, loan amount, employment type, and financial history to predict whether a loan should be approved or rejected. The model helps banks and financial institutions make faster, unbiased, and data-driven decisions, reducing human errors.
By leveraging historical data, it enhances accuracy, efficiency, and fairness in the approval process. The ultimate goal is to streamline operations, minimize manual workload, and improve customer satisfaction through reliable predictions

KEY INSIGHTS

1.Applicants with higher income levels and strong credit histories are more likely to get their loans approved.

2.A high loan amount relative to income or poor credit score reduces approval chances.

3.Employment stability and lower debt-to-income ratios play a major role in loan eligibility.

4.The model achieves high accuracy in predicting approvals, minimizing manual decision errors.

5.Using ML helps speed up processing while ensuring fair and data-driven loan evaluations

SUMMARY

This Loan Approval Prediction project uses machine learning to automate the loan approval process. It analyzes applicant data such as income, credit score, loan amount, and employment type to predict loan eligibility. The goal is to make loan decisions faster, more accurate, and unbiased, reducing manual effort and human error. By using predictive modeling, the system helps financial institutions improve efficiency and fairness in loan processing

RECOMMENDATIONS

1. Integrate real-time data such as updated credit scores and income verification to improve prediction accuracy.
2. Balance the dataset by handling class imbalance to avoid bias toward loan rejections or approvals.
3. Use feature importance analysis to focus on the most impactful factors like income, credit score, and loan amount.
4. Regularly retrain the model with new data to adapt to changing financial and economic trends.
5. Implement a dashboard or alert system for loan officers to interpret predictions transparently.
6. Apply explainable AI (XAI) tools to make the decision making process more understandable and trustworthy.
7. Continuously monitor performance metrics (accuracy, precision, recall) to maintain model reliability

```python
import pandas as pd

df = pd.read_csv(r"C:\Users\parth\OneDrive\Desktop\
loan_approval_dataset.csv")
df.head()
```

```
   loan_id   no_of_dependents      education  self_employed
income_annum  \
0        1                  2       Graduate             No
9600000
1        2                  0   Not Graduate            Yes
4100000
2        3                  3       Graduate             No
9100000
3        4                  3       Graduate             No
8200000
4        5                  5   Not Graduate            Yes
9800000

    loan_amount   loan_term   cibil_score
residential_assets_value  \
0      29900000          12           778                  2400000

1      12200000           8           417                  2700000

2      29700000          20           506                  7100000

3      30700000           8           467                 18200000

4      24200000          20           382                 12400000


    commercial_assets_value   luxury_assets_value
bank_asset_value  \
0                  17600000              22700000                  8000000

1                   2200000               8800000                  3300000

2                   4500000              33300000                 12800000

3                   3300000              23300000                  7900000

4                   8200000              29400000                  5000000


   loan_status
0     Approved
1     Rejected
2     Rejected
3     Rejected
4     Rejected
```

```python
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputefrom sklearn.pipeline import
Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, classification_report,
    roc_auc_score, RocCurveDisplay
)
RND = 42

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier

from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, classification_report, confusion_matrix,
    roc_curve, auc
)

RND = 42

# 2. Clean column names & quick info


# make names: lowercase, strip spaces, replace inner spaces with _
df.columns = [str(c).strip().lower().replace(" ", "_") for c in
df.columns]

print("Columns:", df.columns.tolist())
print("\nData info:")
display(df.info())
```

```
Columns: ['loan_id', 'no_of_dependents', 'education', 'self_employed',
'income_annum', 'loan_amount', 'loan_term', 'cibil_score',
'residential_assets_value', 'commercial_assets_value',
'luxury_assets_value', 'bank_asset_value', 'loan_status']

Data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   loan_id                   4269 non-null   int64 
 1   no_of_dependents          4269 non-null   int64 
 2   education                 4269 non-null   object
 3   self_employed             4269 non-null   object
 4   income_annum              4269 non-null   int64 
 5   loan_amount               4269 non-null   int64 
 6   loan_term                 4269 non-null   int64 
 7   cibil_score               4269 non-null   int64 
 8   residential_assets_value  4269 non-null   int64 
 9   commercial_assets_value   4269 non-null   int64 
 10  luxury_assets_value       4269 non-null   int64 
 11  bank_asset_value          4269 non-null   int64 
 12  loan_status               4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB

None
```

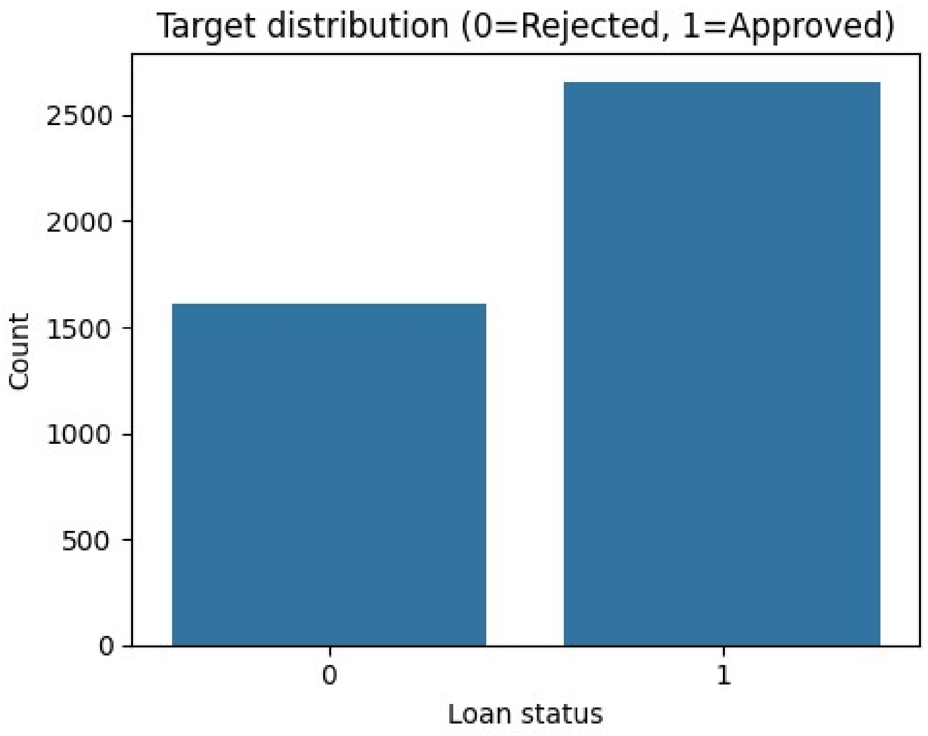
```
#Identify  columns
# our target is 'loan_status', but we make it robust:
candidates = [c for c in df.columns if c in (
    "loan_status", "status", "approved", "approval",
    "loan_decision", "loan_approved"
)]
if not candidates:
    candidates = [c for c in df.columns if "loan" in c and "status" in
c]

if candidates:
    target_col = candidates[0]
else:
    target_col = df.columns[-1]   # fallback: last column
```

```python
print(f"Selected target column: {target_col!r}")
print("Unique target values (sample):", df[target_col].unique()[:10])

# --- map Approved / Rejected etc. to binary ----
y_raw = df[target_col].astype(str).str.strip().str.lower()

def map_to_binary(s):
    if pd.isna(s):
        return np.nan
    if s in {"y", "yes", "approved", "accept", "accepted", "1",
"true", "t"} or "approve" in s:
        return 1
    if s in {"n", "no", "rejected", "reject", "0", "false", "f",
"deny", "denied"} or "reject" in s:
        return 0
    # if numeric 0/1
    try:
        v = float(s)
        if v == 1.0: return 1
        if v == 0.0: return 0
    except Exception:
        pass
    # default guess
    if "yes" in s or "approve" in s:
        return 1
    return 0

y = y_raw.apply(map_to_binary)

print("\nTarget distribution after mapping:")
display(y.value_counts(dropna=False))

# drop rows with missing target (should be none)
mask_na = y.isna()
if mask_na.sum() > 0:
    print(f"Dropping {mask_na.sum()} rows with missing target")
    df = df.loc[~mask_na].reset_index(drop=True)
    y = y.loc[~mask_na].reset_index(drop=True)
else:
    y = y.reset_index(drop=True)
```

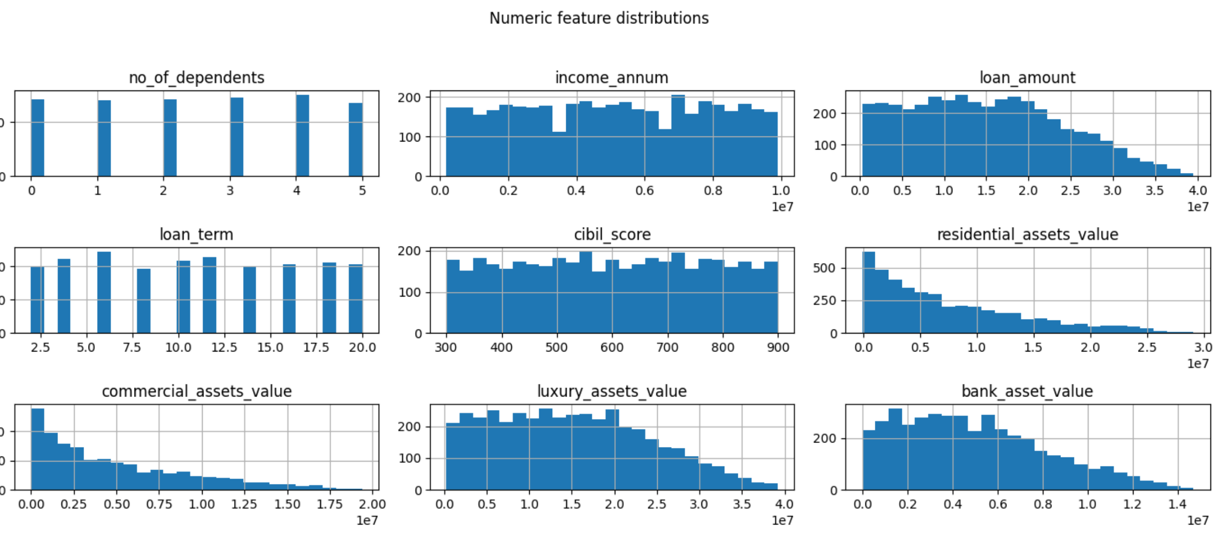
```
Selected target column: 'loan_status'
Unique target values (sample): [' Approved' ' Rejected']

Target distribution after mapping:

loan_status
1    2656
0    1613
Name: count, dtype: int64
```

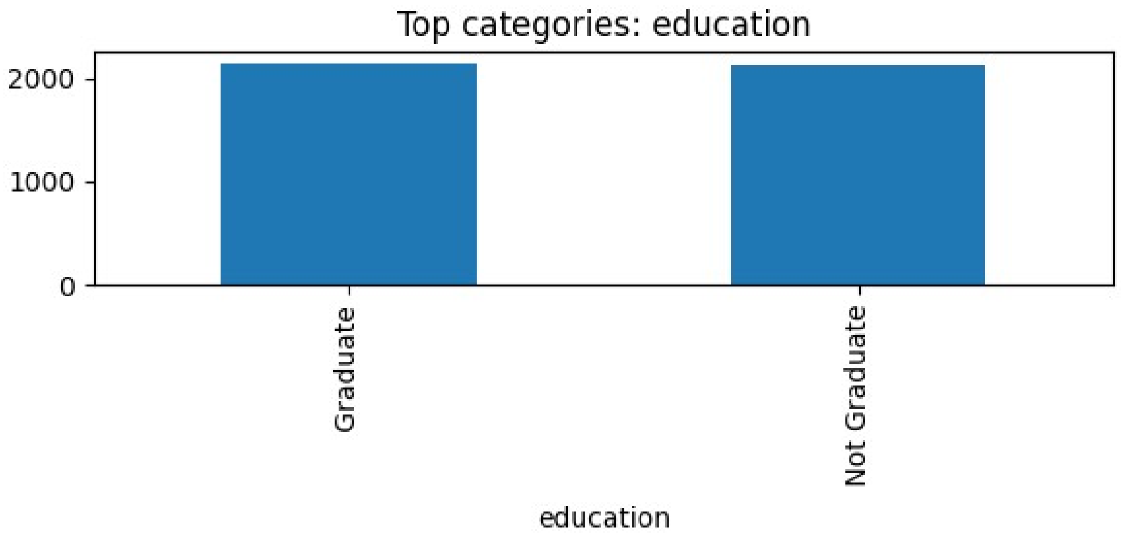
```
# Quick EDA & Visualizations
from IPython.display import display

# Missing values
missing = df.isnull().sum().sort_values(ascending=False)
print("Columns with missing values:")
display(missing[missing > 0])

if (missing > 0).any():
    plt.figure(figsize=(10,4))
    missing[missing > 0].plot(kind="bar")
    plt.title("Missing values per column")
    plt.ylabel("Count")
    plt.tight_layout()
    plt.show()
else:
    print("□ No missing values found. Skipping missing-value plot.")

# Class balance
plt.figure(figsize=(5,4))
sns.countplot(x=pd.Series(y))
plt.title("Target distribution (0=Rejected, 1=Approved)")
plt.xlabel("Loan status")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# numeric / categorical columns (excluding id)
id_like = [c for c in df.columns if c.endswith("_id") or c ==
"loan_id" or c.startswith("id")]
print("ID-like columns (will be dropped later):", id_like)

numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
numeric_cols = [c for c in numeric_cols if c not in id_like]
categorical_cols = df.select_dtypes(include=["object", "category",
"bool"]).columns.tolist()
categorical_cols = [c for c in categorical_cols if c != target_col]

print("Numeric cols:", numeric_cols)
print("Categorical cols:", categorical_cols)

# Histograms for numeric
if numeric_cols:
    df[numeric_cols].hist(bins=25, figsize=(14,6))
    plt.suptitle("Numeric feature distributions")
    plt.tight_layout(rect=[0,0,1,0.95])
    plt.show()

# Bar charts for categoricals
for c in categorical_cols:
    plt.figure(figsize=(6,3))
    df[c].value_counts().plot(kind="bar")
    plt.title(f"Top categories: {c}")
    plt.tight_layout()
    plt.show()

# Correlation heatmap (numeric + target)
corr_df = df[numeric_cols].copy()
corr_df["__target__"] = y.values.astype(float)

plt.figure(figsize=(10,8))
sns.heatmap(corr_df.corr(), annot=False, cmap="coolwarm", center=0)
plt.title("Correlation matrix (numeric features + target)")
plt.tight_layout()
plt.show()
```

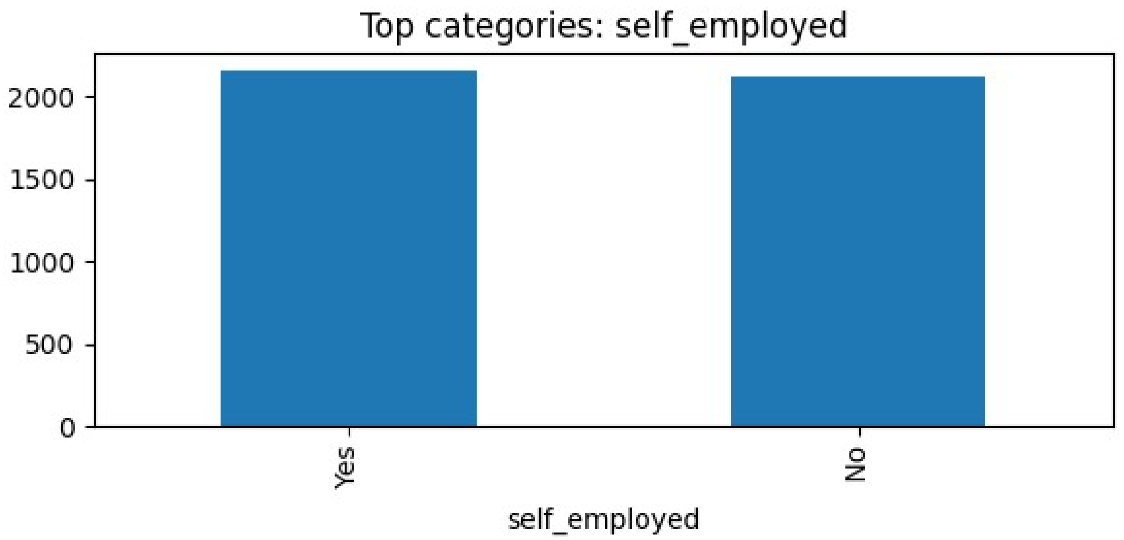
```
Columns with missing values:

Series([], dtype: int64)

 No missing values found. Skipping missing-value plot.
```

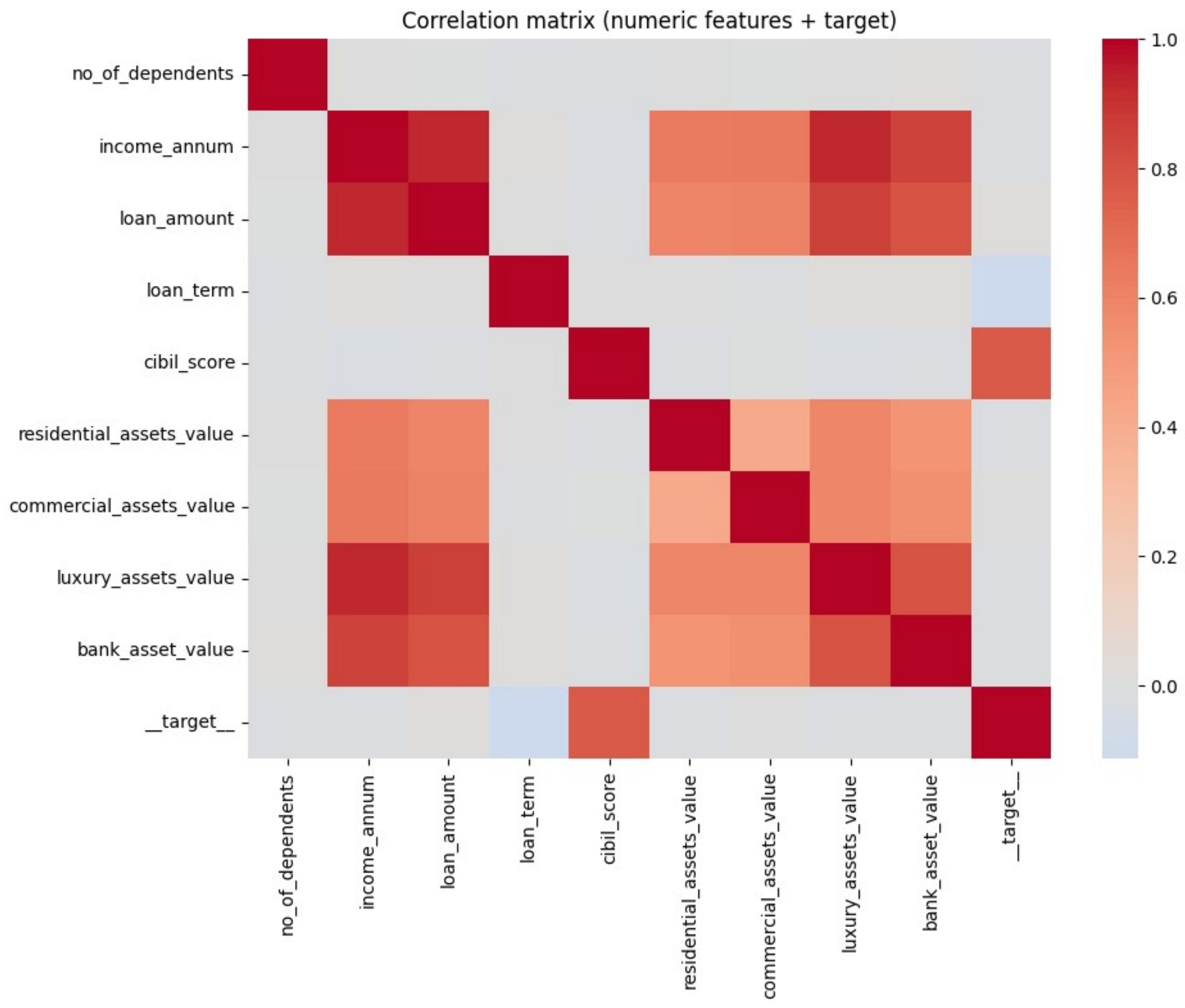


```
ID-like columns (will be dropped later): ['loan_id']
Numeric cols: ['no_of_dependents', 'income_annum', 'loan_amount',
'loan_term', 'cibil_score', 'residential_assets_value',
'commercial_assets_value', 'luxury_assets_value', 'bank_asset_value']
Categorical cols: ['education', 'self_employed']
```


Numeric feature distributions
no_of_dependents
income_annum
loan_amount
loan_term
cibil_score
residential_assets_value
commercial_assets_value
luxury_assets_value
bank_asset_value
1e7


Top categories: education
Graduate
Not Graduate
education


Top categories: self_employed
Yes
No
self_employed



```
#  Train–Test Split
X = df.drop(columns=[target_col], errors="ignore")
y = y.astype(int)

# drop id columns
for c in id_like:
    if c in X.columns:
        X = X.drop(columns=c)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RND, stratify=y
)

print("Train shape:", X_train.shape)
print("Test shape :", X_test.shape)
```

```
Train shape: (3415, 11)
Test shape : (854, 11)
```

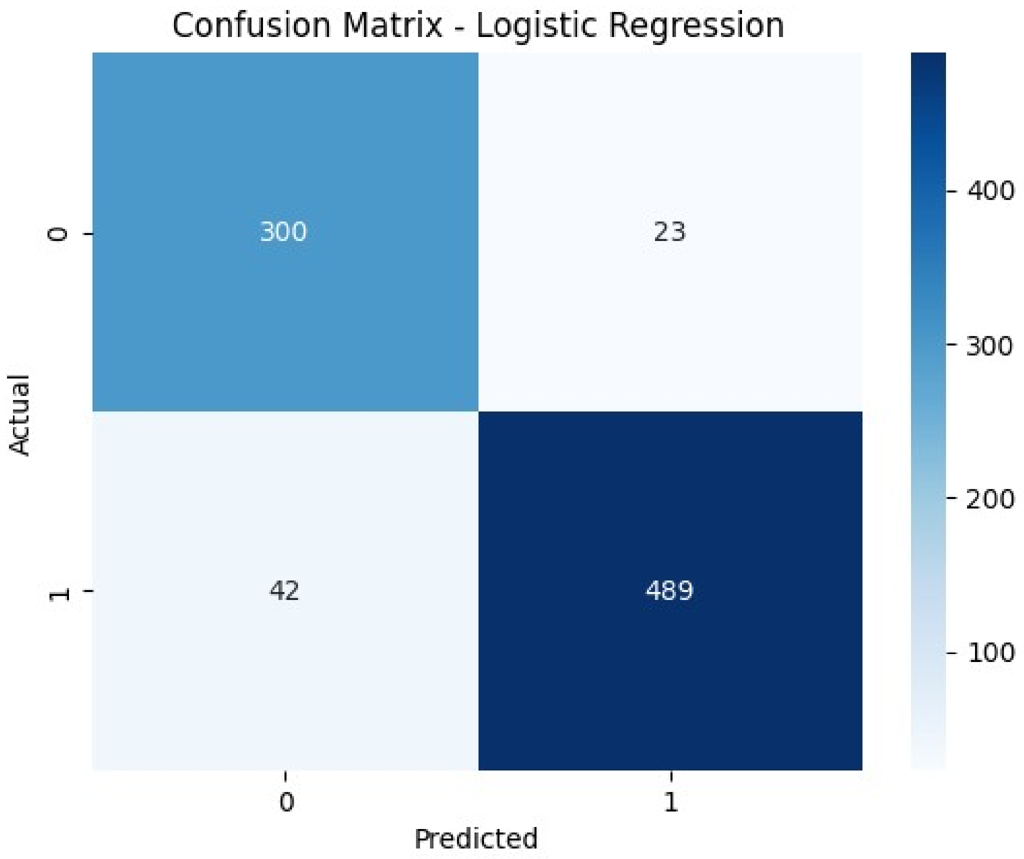
```
# Pre-processing (ColumnTransformer)
num_cols = X.select_dtypes(include=np.number).columns
cat_cols = X.select_dtypes(include=["object", "category",
"bool"]).columns

print("Numeric columns used:", list(num_cols))
print("Categorical columns used:", list(cat_cols))

numeric_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# NOTE:
# then change sparse_output=False
categorical_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore",
sparse_output=False))
])

preprocessor = ColumnTransformer([
    ("num", numeric_pipe, num_cols),
    ("cat", categorical_pipe, cat_cols)
])
```

```
Numeric columns used: ['no_of_dependents', 'income_annum',
'loan_amount', 'loan_term', 'cibil_score', 'residential_assets_value',
'commercial_assets_value', 'luxury_assets_value', 'bank_asset_value']
Categorical columns used: ['education', 'self_employed']
```

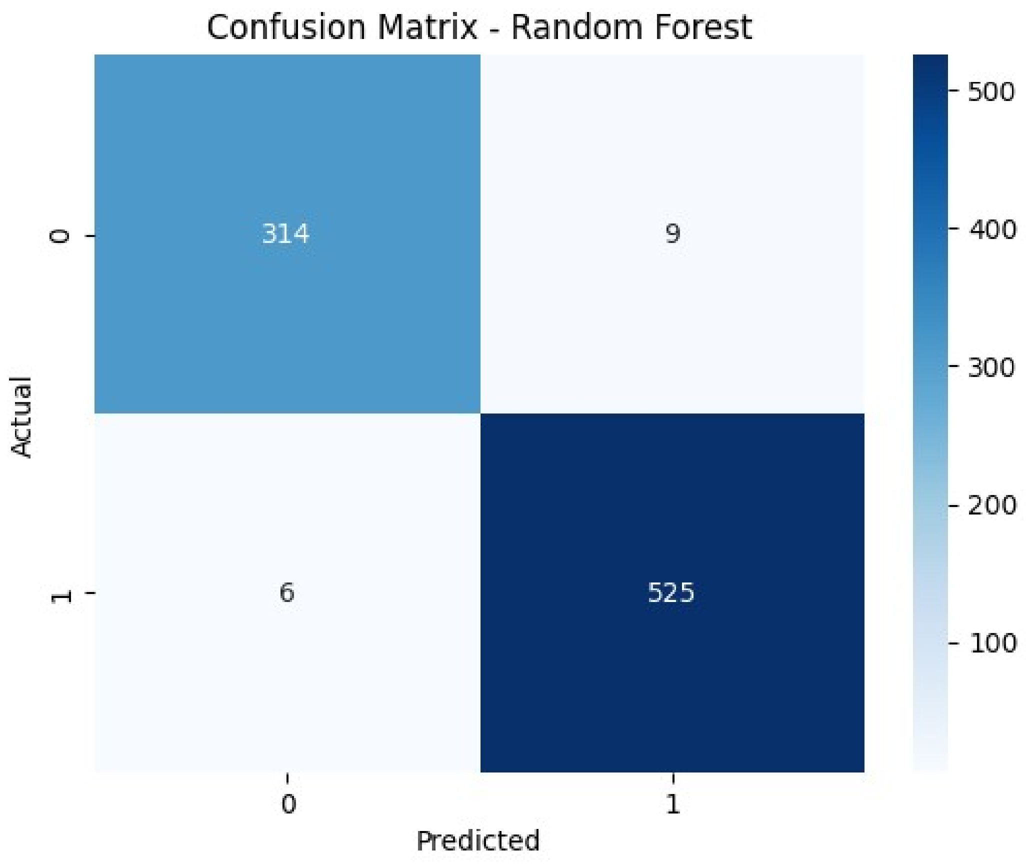
```
# 7. Model Training & Evaluation

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000,
class_weight="balanced"),
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=RND, class_weight="balanced"),
    "Gradient Boosting": GradientBoostingClassifier(random_state=RND)
}

roc_curves = {}

for name, model in models.items():
    pipe = Pipeline([("prep", preprocessor), ("model", model)])
    pipe.fit(X_train, y_train)

    preds = pipe.predict(X_test)
    proba = pipe.predict_proba(X_test)[:, 1] if hasattr(pipe,
"predict_proba") else None
```

```
    print(f"\n{name} Results:")
    print(classification_report(y_test, preds))

    # Confusion matrix
    sns.heatmap(confusion_matrix(y_test, preds), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix - {name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    if proba is not None:
        fpr, tpr, _ = roc_curve(y_test, proba)
        roc_auc = auc(fpr, tpr)
        roc_curves[name] = (fpr, tpr, roc_auc)

# ROC curves plot
plt.figure(figsize=(6,5))
for name, (fpr, tpr, roc_auc) in roc_curves.items():
    plt.plot(fpr, tpr, label=f"{name} (AUC={roc_auc:.2f})")

plt.plot([0,1], [0,1], "k--", label="Random guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.legend()
plt.tight_layout()
plt.show()
```

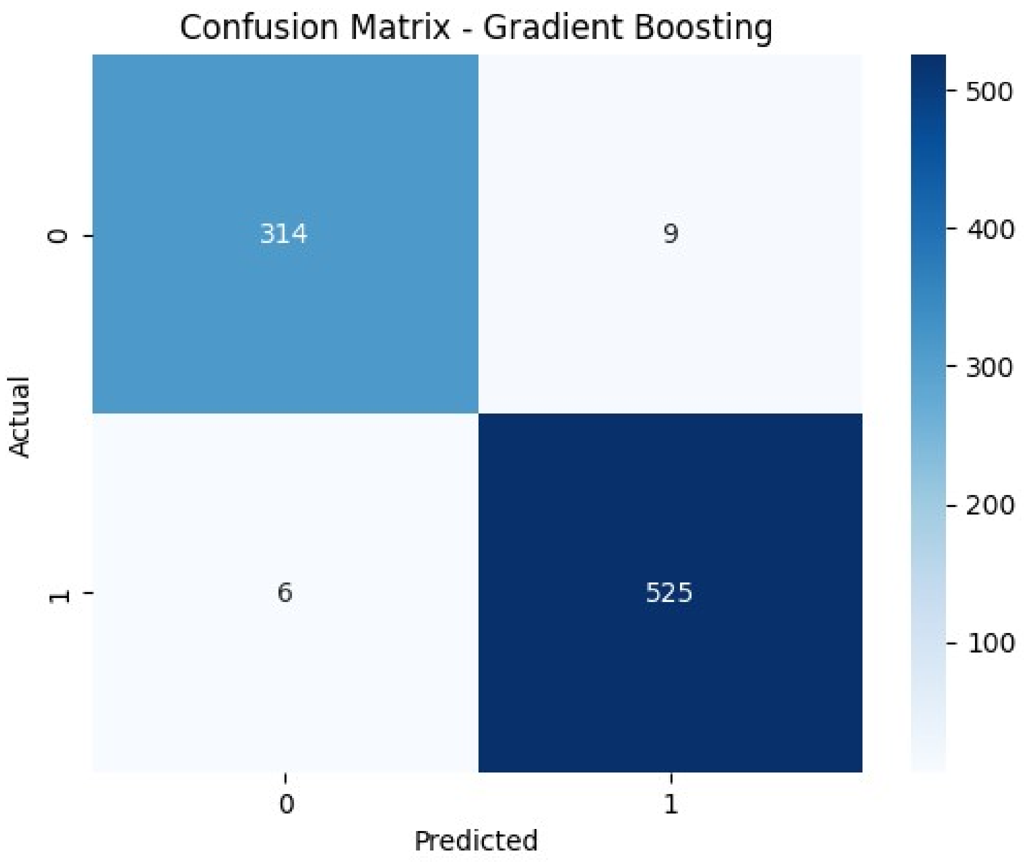
```
Logistic Regression Results:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       323
           1       0.96      0.92      0.94       531

    accuracy                           0.92       854
   macro avg       0.92      0.92      0.92       854
weighted avg       0.93      0.92      0.92       854
```

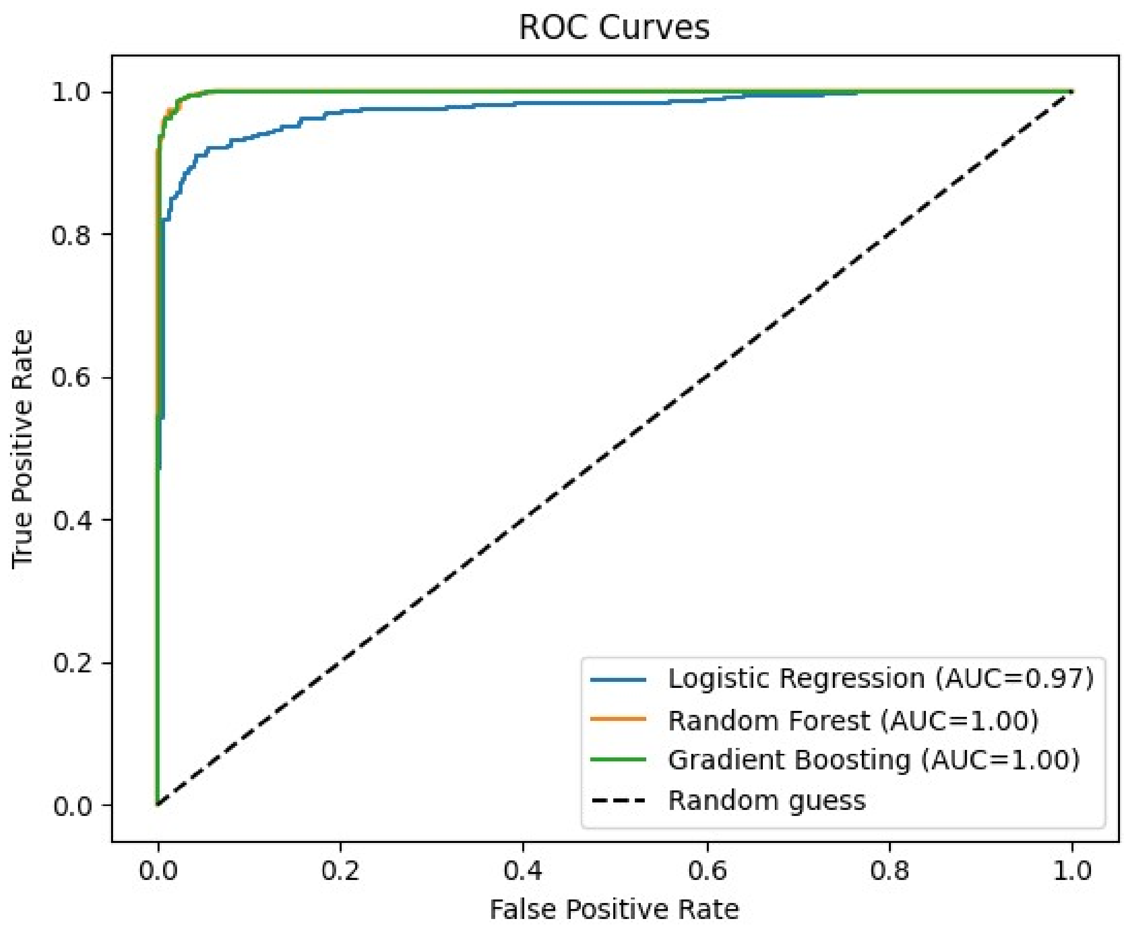


```
Random Forest Results:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       323
           1       0.98      0.99      0.99       531

    accuracy                           0.98       854
   macro avg       0.98      0.98      0.98       854
weighted avg       0.98      0.98      0.98       854
```



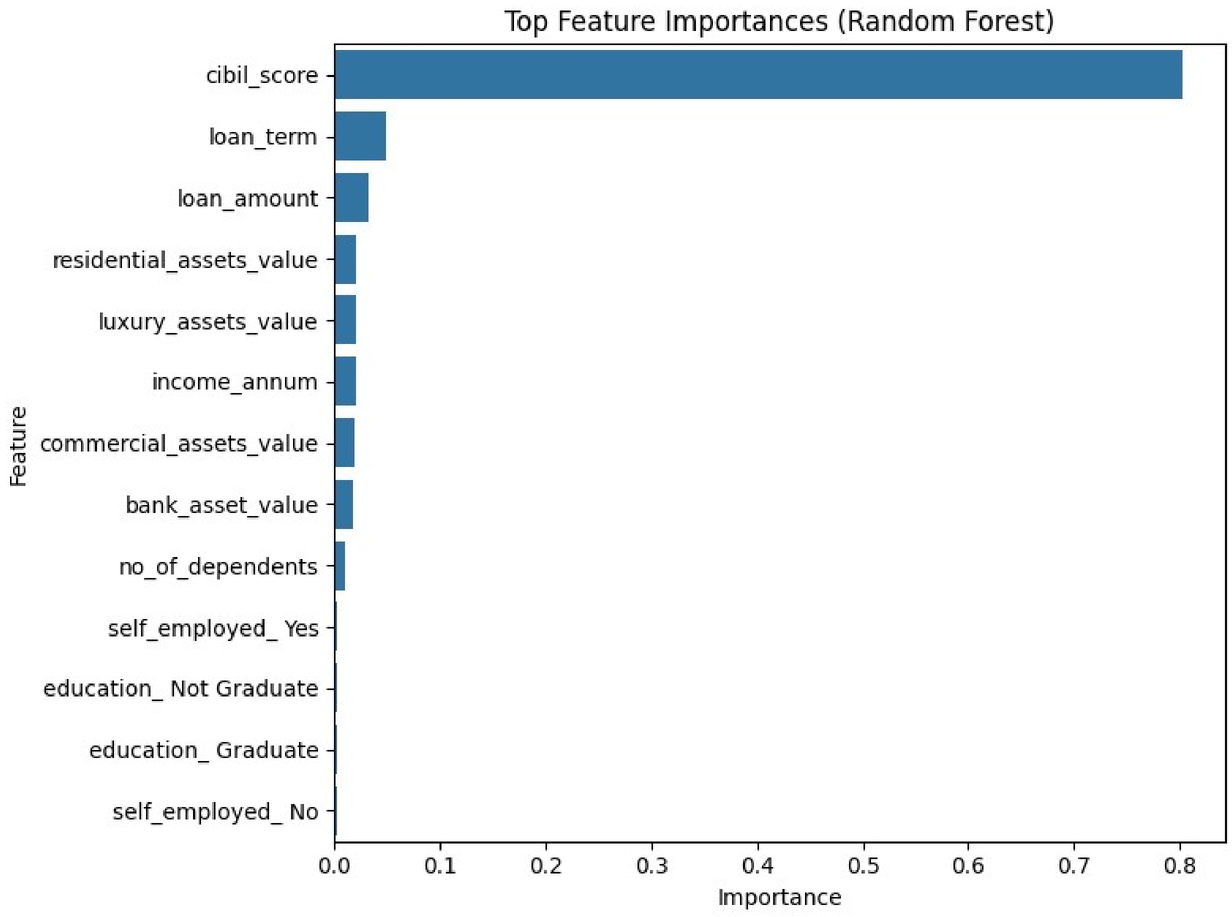
```
Gradient Boosting Results:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       323
           1       0.98      0.99      0.99       531

    accuracy                           0.98       854
   macro avg       0.98      0.98      0.98       854
weighted avg       0.98      0.98      0.98       854
```


Confusion Matrix - Gradient Boosting
Actual
Predicted
0
1
314
9
6
525
500
400
300
200
100



```
# ==========
# 8. Feature Importance (Random Forest)


best_model = RandomForestClassifier(
    n_estimators=200, random_state=RND, class_weight="balanced"
)
pipe_best = Pipeline([("prep", preprocessor), ("model", best_model)])
pipe_best.fit(X_train, y_train)

# need fitted preprocessor to get OHE feature names
preprocessor.fit(X_train)

# numeric + one-hot encoded categorical feature names
num_feature_names = list(num_cols)
ohe = preprocessor.named_transformers_["cat"].named_steps["onehot"]
cat_feature_names = ohe.get_feature_names_out(cat_cols)
all_feature_names = num_feature_names + list(cat_feature_names)

importances = pipe_best.named_steps["model"].feature_importances_
```

```python
fi = pd.DataFrame({"Feature": all_feature_names, "Importance":
importances})
fi = fi.sort_values("Importance", ascending=False).head(15)

plt.figure(figsize=(8,6))
sns.barplot(x="Importance", y="Feature", data=fi)
plt.title("Top Feature Importances (Random Forest)")
plt.tight_layout()
plt.show()
```


Top Feature Importances (Random Forest)
cibil_score
loan_term
loan_amount
residential_assets_value
luxury_assets_value
income_annum
commercial_assets_value
bank_asset_value
no_of_dependents
self_employed_ Yes
education_ Not Graduate
education_ Graduate
self_employed_ No
Feature
Importance