```
#  Setup & imports ---
import warnings
warnings.filterwarnings("ignore")

import os
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
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, precision_recall_curve)
import joblib

RND = 42
OUT_DIR = Path("/mnt/data")
OUT_DIR.mkdir(parents=True, exist_ok=True)

import pandas as pd
from pathlib import Path

# Full path to the CSV file
DATA_PATH = Path("F:/C:/Windows/System32/cmd.exe")

# Fallback (optional, in case primary path fails)
if not DATA_PATH.exists():
    DATA_PATH = Path("loan_approval_dataset.csv")

# Check if file exists
if not DATA_PATH.exists():
    raise FileNotFoundError(f"Dataset not found at: {DATA_PATH}")

# Load the dataset
df = pd.read_csv(DATA_PATH)
print("Loaded dataset shape:", df.shape)

# Show first few rows (use print if not in Jupyter)
```

```
try:
    display(df.head())
except NameError:
    print(df.head())
```

```
Loaded dataset shape: (4269, 13)
```

```
   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
```

```
#  2. Clean column names & quick info ---
# Normalize column names - strip spaces, lower, replace spaces with
underscores
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
```

```
# 3. Identify target column robustly and map to 0/1 ---
# Common target names to search
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:
    # fallback to last column
    target_col = df.columns[-1]

print(f"Selected target column: '{target_col}'")
print("Unique target values (sample):", df[target_col].unique()[:20])

# Map many common variants 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
    # fallback: if looks numeric 0/1
    try:
        v = float(s)
        if v == 1.0:
            return 1
        if v == 0.0:
            return 0
    except Exception:
        pass
    # default: try contains 'yes' or 'approve'
    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 where target is missing (NaN)
na_mask = y.isna()
if na_mask.sum() > 0:
    print(f"Dropping {na_mask.sum()} rows with missing target")
    df = df.loc[~na_mask].reset_index(drop=True)
    y = y.loc[~na_mask].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:
```

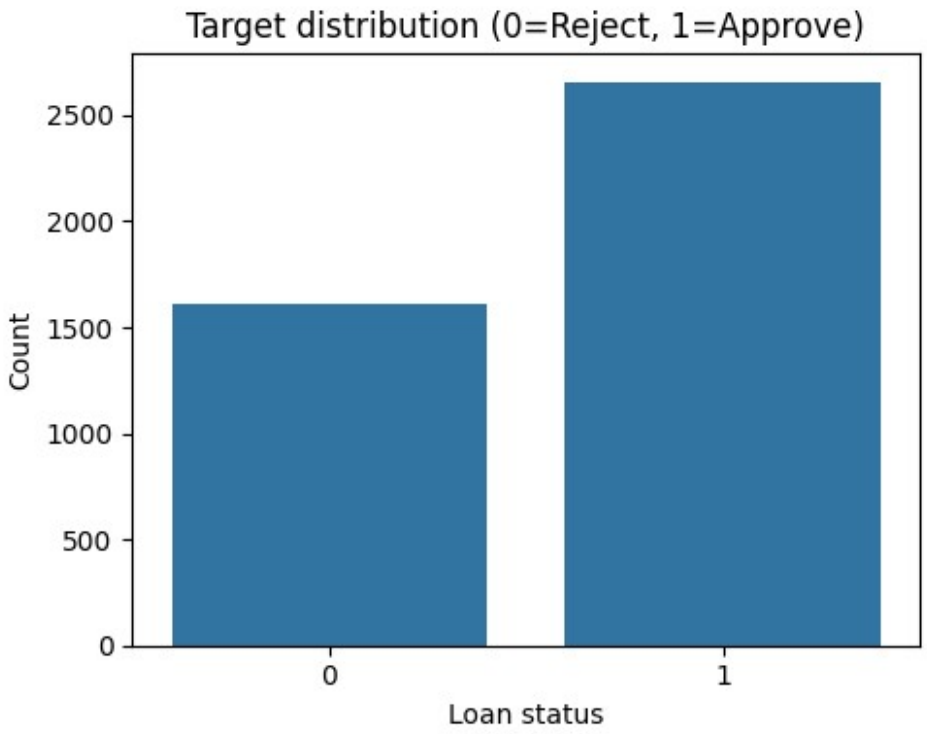
```
loan_status
1    2656
0    1613
Name: count, dtype: int64
```

```
# 4. Quick EDA & visualizations (robust/fixed) ---
from IPython.display import display

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

missing_nonzero = missing[missing > 0]  # only columns that actually
have missing values

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

# show class balance (safe-guard if y empty)
if y is not None and len(y) > 0:
    plt.figure(figsize=(5,4))
    # ensure y is a Series (works if y is a numpy array too)
    sns.countplot(x=pd.Series(y))
    plt.title("Target distribution (0=Reject, 1=Approve)")
    plt.xlabel("Loan status")
    plt.ylabel("Count")
    plt.tight_layout()
    plt.show()
else:
    print("Warning: target series 'y' is empty. Skipping class balance
plot.")

# numeric vs categorical split (after removing obvious ID columns)
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):", 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 and c !=
getattr(y, 'name', None)]
categorical_cols = df.select_dtypes(include=['object', 'category',
```

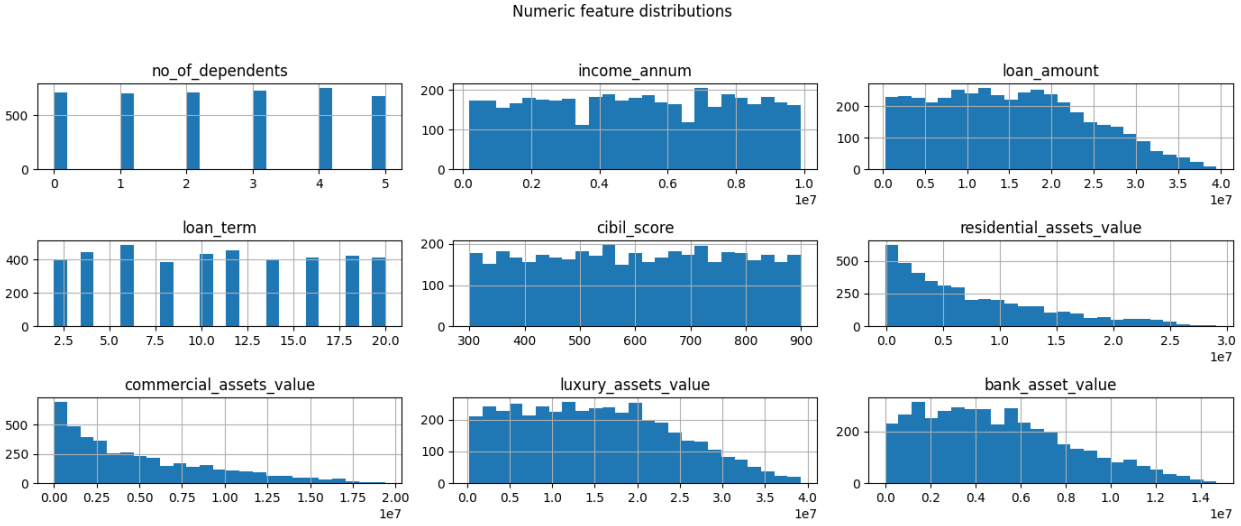
```
'bool']).columns.tolist()
categorical_cols = [c for c in categorical_cols if c != target_col]

print("\nNumeric cols:", numeric_cols)
print("Categorical cols:", categorical_cols)

# histograms for numeric features (guarded)
if numeric_cols:
    try:
        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()
    except Exception as e:
        print("Could not plot numeric histograms:", e)
else:
    print("No numeric columns to plot histograms for.")

# top categories for categorical features (bar plots, guarded)
if categorical_cols:
    for c in categorical_cols:
        plt.figure(figsize=(6,3))
        vc = df[c].fillna("MISSING").value_counts().nlargest(10)
        if vc.shape[0] > 0:
            vc.plot(kind="bar")
            plt.title(f"Top categories: {c}")
            plt.tight_layout()
            plt.show()
        else:
            print(f"No categories to plot for column: {c}")
else:
    print("No categorical columns to plot.")

# correlation heatmap for numeric features (including mapped target) -
guarded
if numeric_cols:
    try:
        corr_df = df[numeric_cols].copy()
        # add target as numeric for correlation if not present
        corr_df["__target__"] = pd.Series(y).astype(float).values
        corr = corr_df.corr()
        if corr.shape[0] > 0 and corr.shape[1] > 0:
            plt.figure(figsize=(10,8))
            sns.heatmap(corr, annot=False, cmap="coolwarm", center=0)
            plt.title("Correlation matrix (numeric features +
target)")
            plt.tight_layout()
            plt.show()
        else:
            print("Correlation matrix is empty; skipping heatmap.")
```

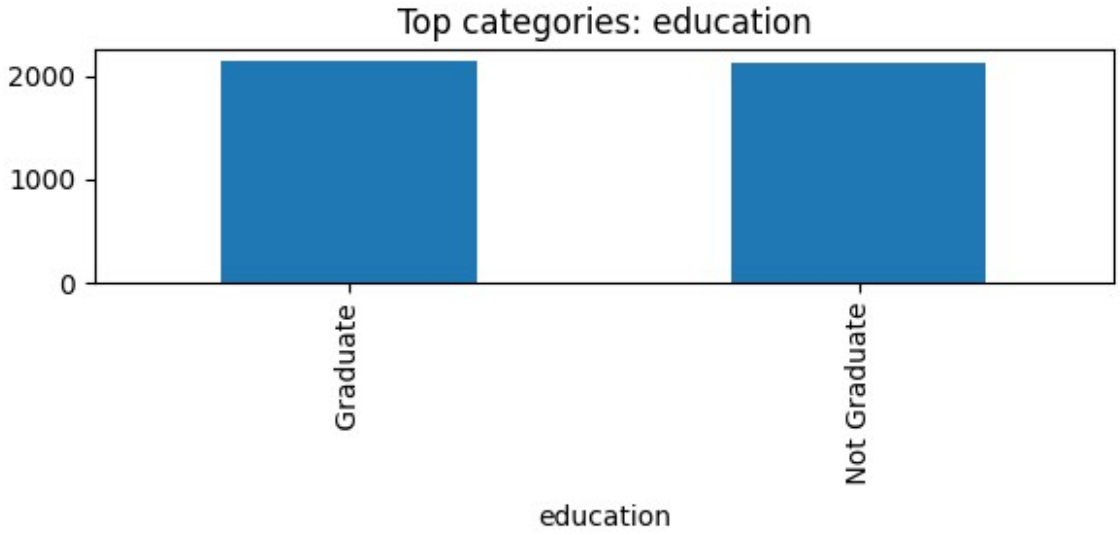
```
    except Exception as e:
        print("Could not compute/plot correlation heatmap:", e)
else:
    print("No numeric columns available for correlation heatmap.")
```

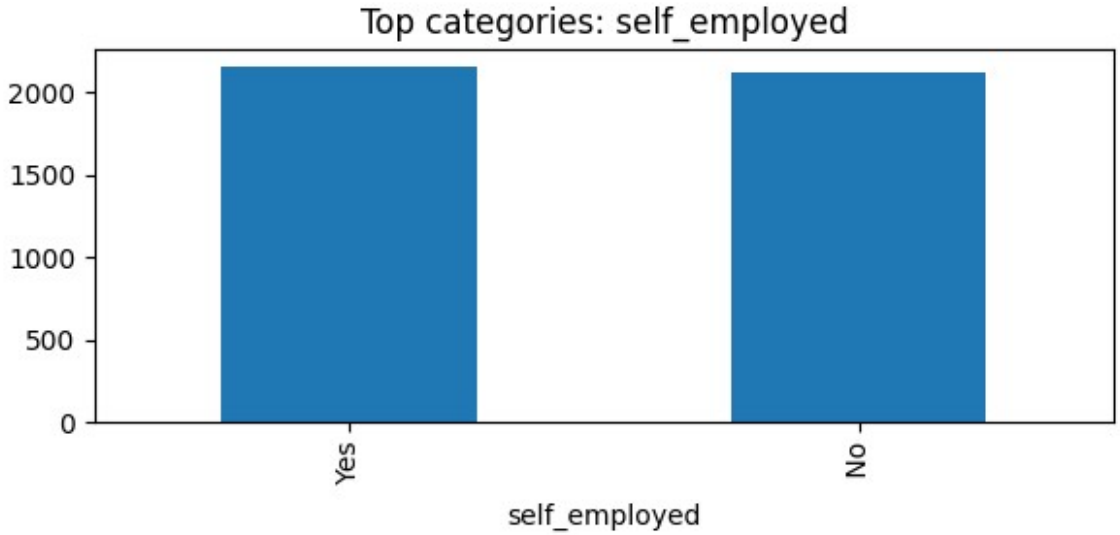
```
Columns with missing values:

Series([], dtype: int64)

 No missing values found in any column. (Skipping missing-values bar plot.)
```

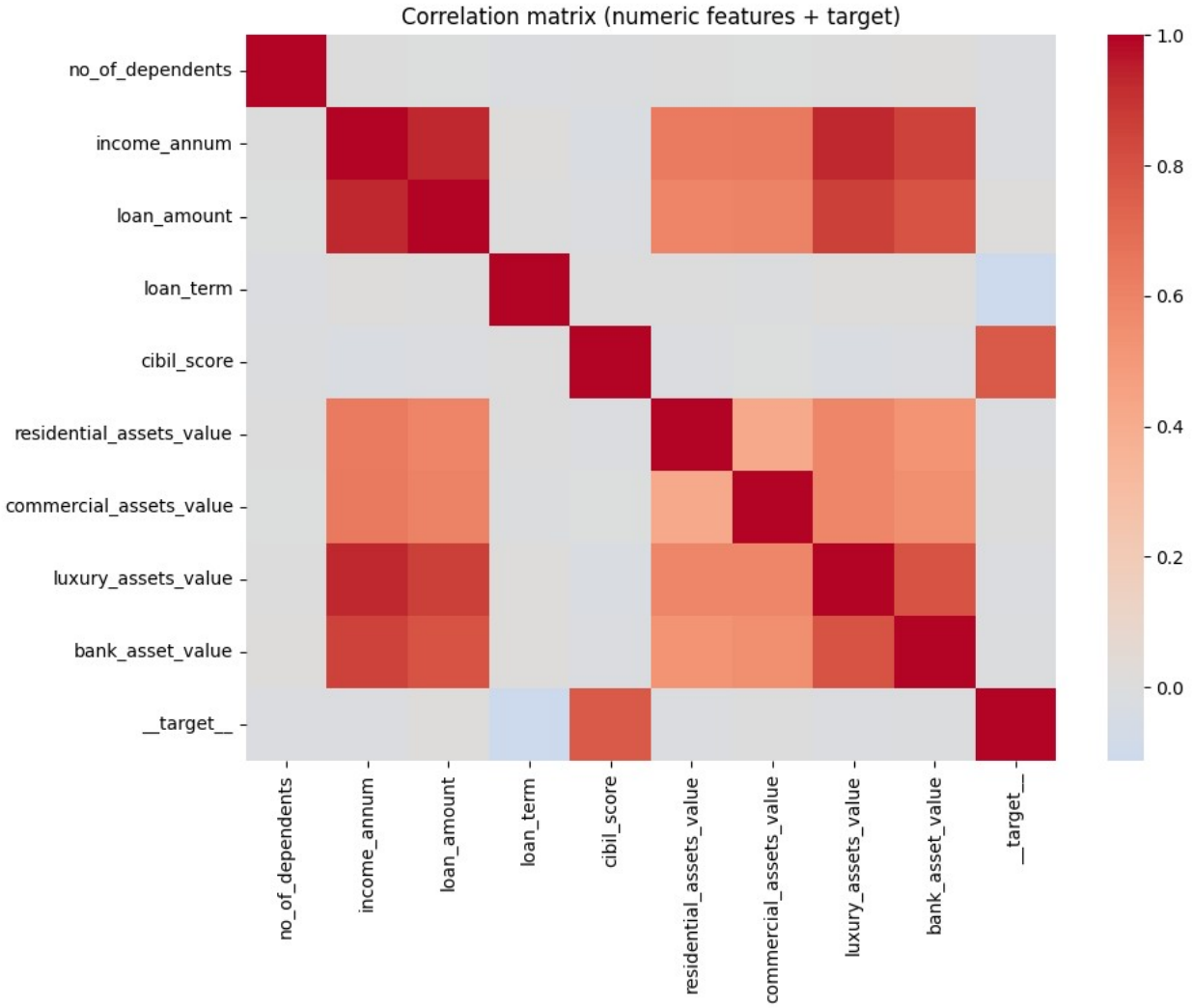



```
ID-like columns (will be dropped): ['loan_id']

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

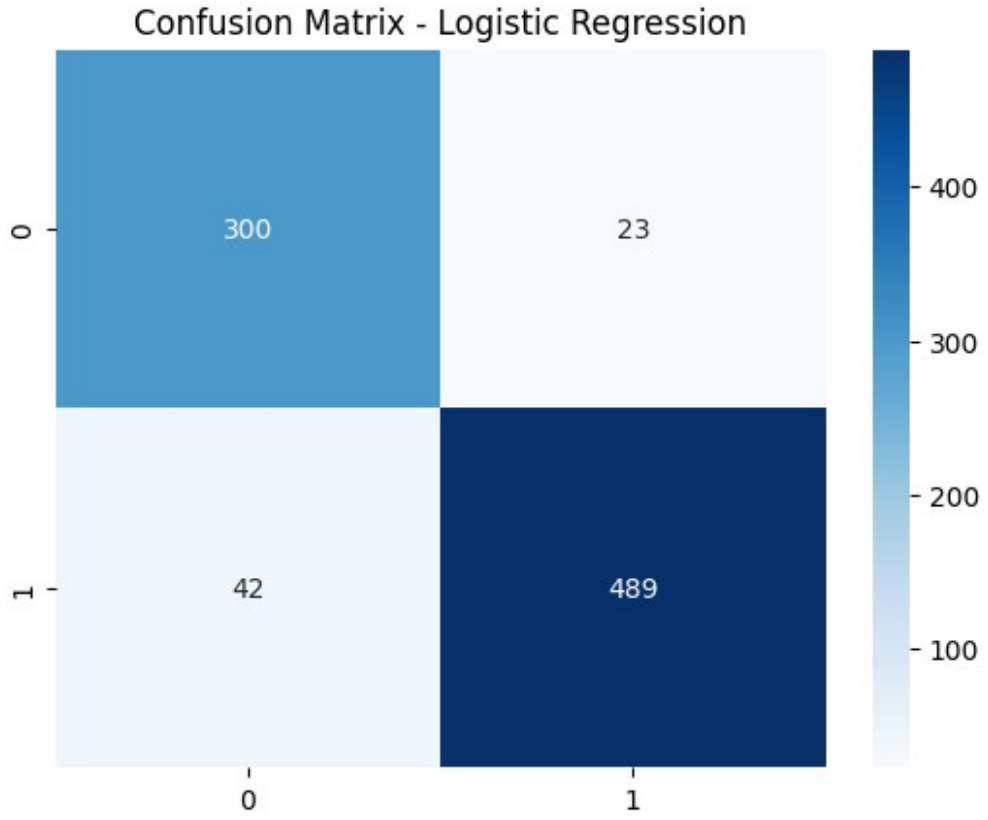

Top categories: self_employed
Yes
No
self_employed



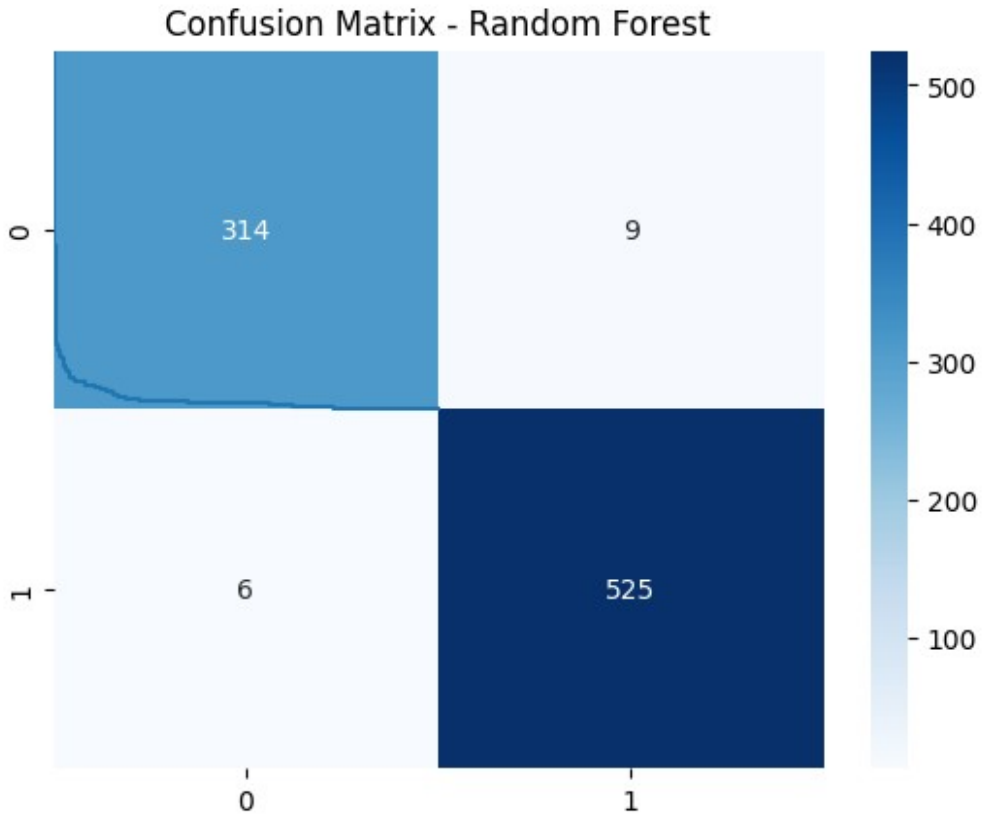

```
# 5 Data Split
X = df.drop(columns=[target_col], errors='ignore')
y = pd.Series(y).astype(int)

# Drop ID columns if present
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=42, stratify=y
)

# 5. Preprocessing
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
```

```
# Identify numeric and categorical columns
num_cols = X.select_dtypes(include=np.number).columns
cat_cols = X.select_dtypes(include=['object', 'category',
'bool']).columns

# Numeric data pipeline
numeric_pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Categorical data pipeline (fixed 'sparse' -> 'sparse_output')
categorical_pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore',
sparse_output=False))
])

# Combine both into a column transformer
preprocessor = ColumnTransformer([
    ('num', numeric_pipe, num_cols),
    ('cat', categorical_pipe, cat_cols)
])

# 6. Model Training
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000,
class_weight='balanced'),
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42, class_weight='balanced'),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42)
}

results = []

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))
    sns.heatmap(confusion_matrix(y_test, preds), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix - {name}")
    plt.show()
```

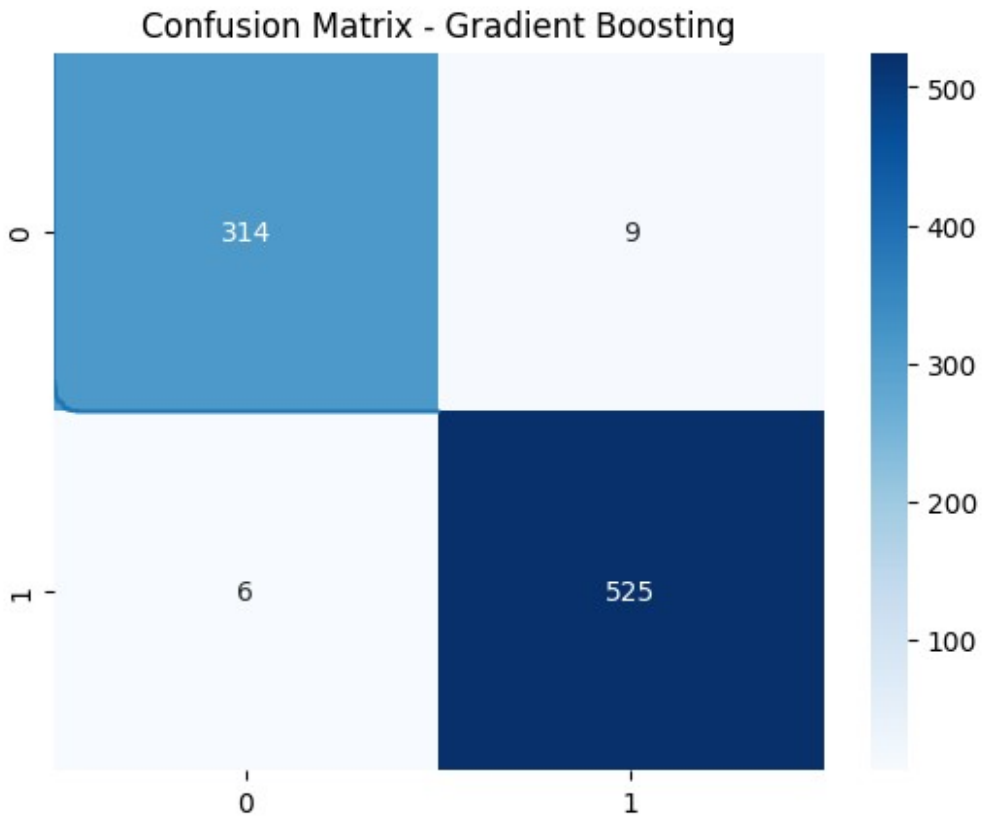
```
    if proba is not None:
        fpr, tpr, _ = roc_curve(y_test, proba)
        plt.plot(fpr, tpr, label=f"{name} (AUC={auc(fpr,tpr):.2f})")

plt.plot([0,1],[0,1],'--', color='gray')
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
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
```

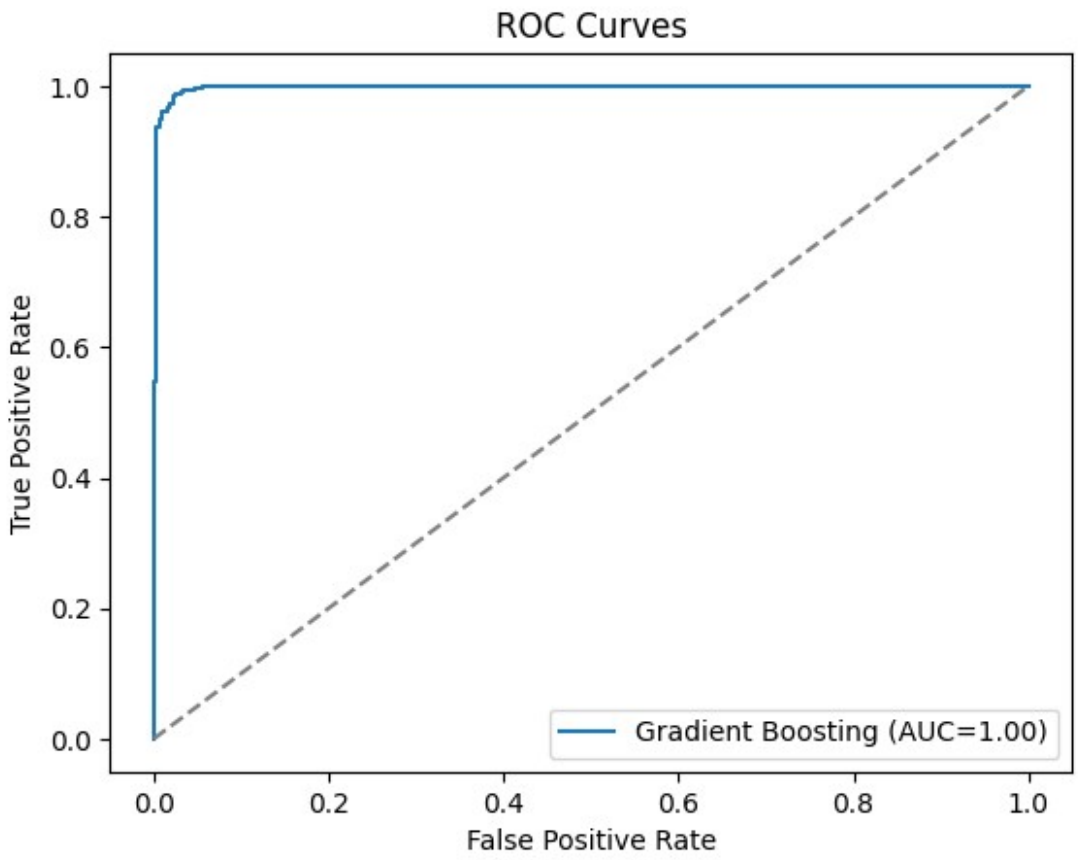


```
🌲 Random Forest Results:
              precision    recall  f1-score   support

           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
```

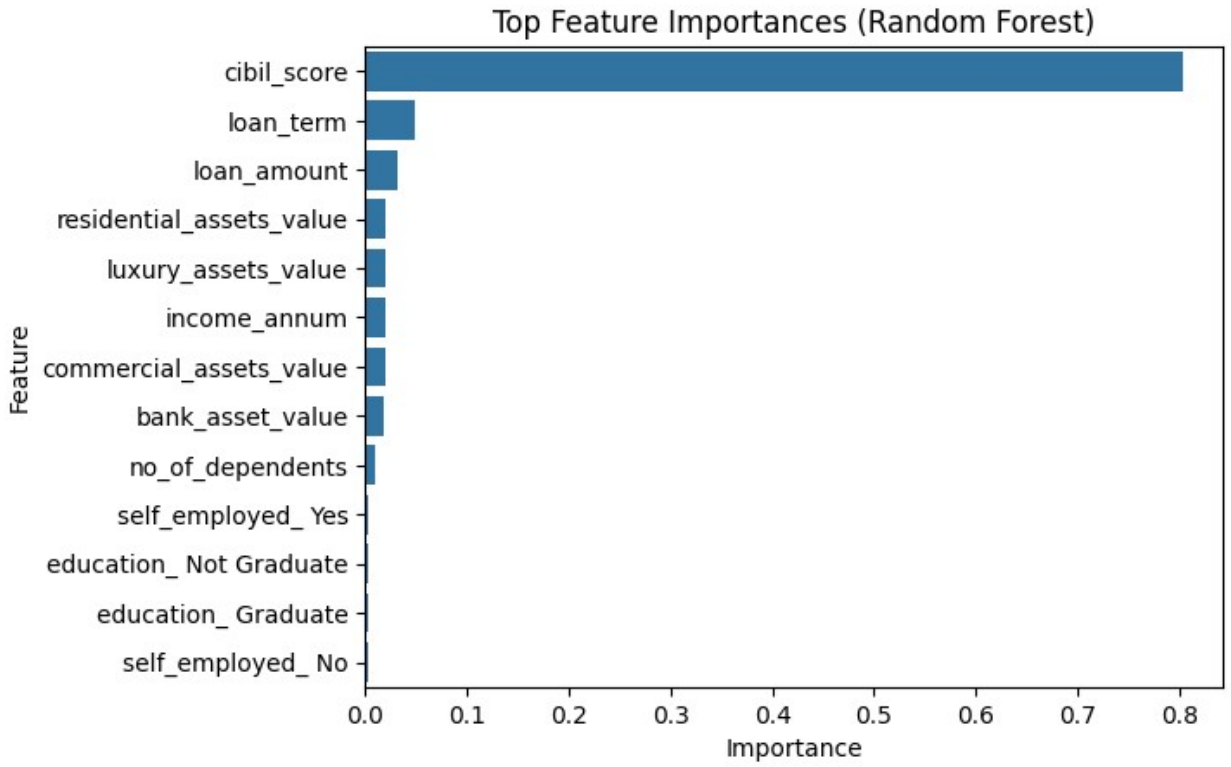



```
🚀 Gradient Boosting Results:
              precision    recall  f1-score   support

           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
314
9
6
525
0
1
0
1
500
400
300
200
100




```
# 7. Feature Importance
best_model = models["Random Forest"]
pipe = Pipeline([('prep', preprocessor), ('model', best_model)])
pipe.fit(X_train, y_train)

preprocessor.fit(X_train)
feat_names = list(num_cols) +
list(preprocessor.named_transformers_['cat']
                                    .named_steps['onehot'].get_feature_
names_out(cat_cols))
importances = pipe.named_steps['model'].feature_importances_
fi = pd.DataFrame({'Feature': feat_names, 'Importance': importances})
fi = fi.sort_values('Importance', ascending=False).head(15)

sns.barplot(x='Importance', y='Feature', data=fi)
plt.title("Top Feature Importances (Random Forest)")
plt.show()
```




```
#8. Save Final Model ---
joblib.dump(pipe, 'loan_approval_model.pkl')
print("□ Model saved as 'loan_approval_model.pkl'")
```

```
□ Model saved as 'loan_approval_model.pkl'
```