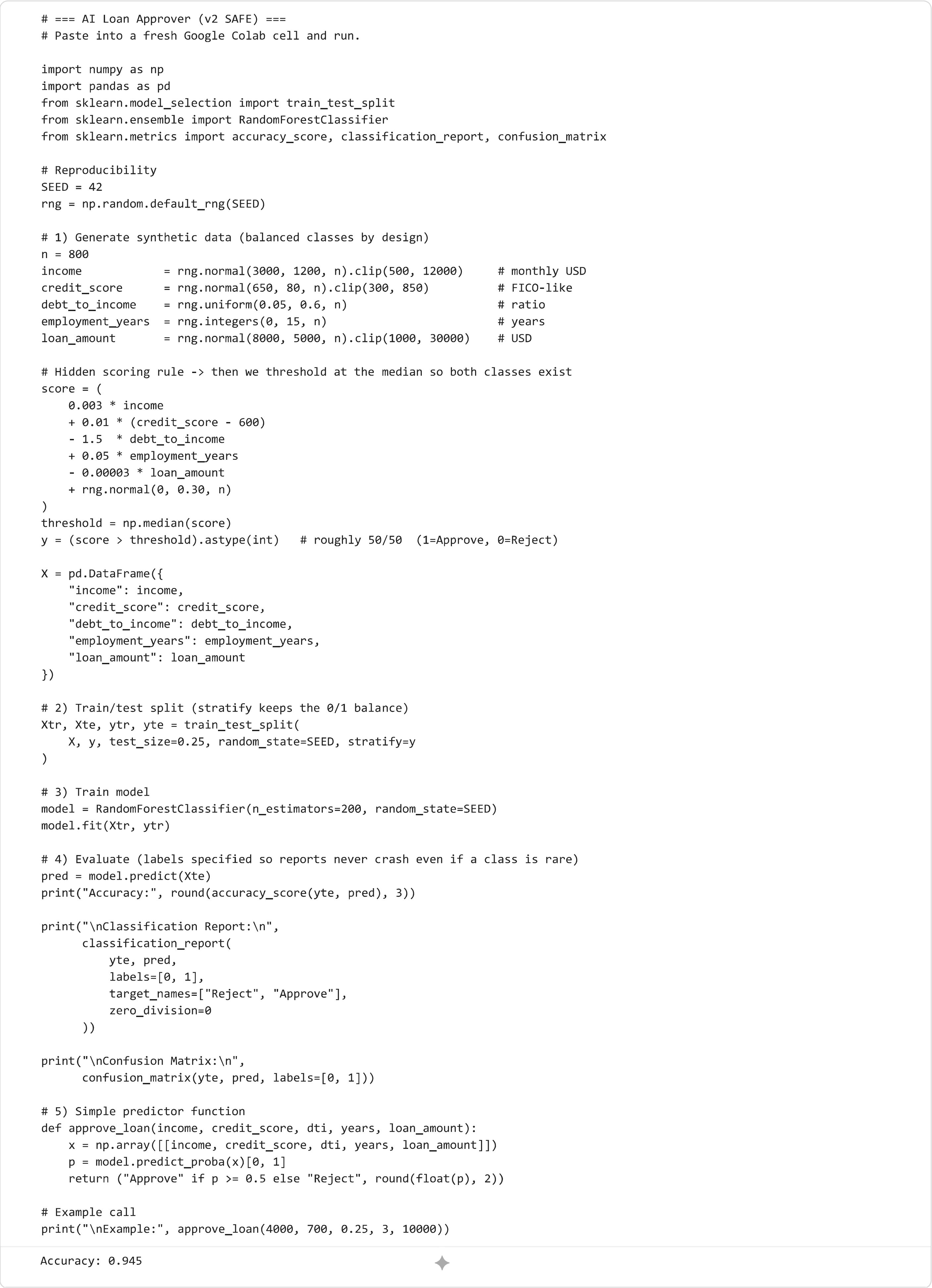

```
# === AI Loan Approver (v2 SAFE) ===
# Paste into a fresh Google Colab cell and run.

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Reproducibility
SEED = 42
rng = np.random.default_rng(SEED)

# 1) Generate synthetic data (balanced classes by design)
n = 800
income            = rng.normal(3000, 1200, n).clip(500, 12000)     # monthly USD
credit_score      = rng.normal(650, 80, n).clip(300, 850)          # FICO-like
debt_to_income    = rng.uniform(0.05, 0.6, n)                      # ratio
employment_years  = rng.integers(0, 15, n)                         # years
loan_amount       = rng.normal(8000, 5000, n).clip(1000, 30000)    # USD

# Hidden scoring rule -> then we threshold at the median so both classes exist
score = (
    0.003 * income
    + 0.01 * (credit_score - 600)
    - 1.5  * debt_to_income
    + 0.05 * employment_years
    - 0.00003 * loan_amount
    + rng.normal(0, 0.30, n)
)
threshold = np.median(score)
y = (score > threshold).astype(int)   # roughly 50/50  (1=Approve, 0=Reject)

X = pd.DataFrame({
    "income": income,
    "credit_score": credit_score,
    "debt_to_income": debt_to_income,
    "employment_years": employment_years,
    "loan_amount": loan_amount
})

# 2) Train/test split (stratify keeps the 0/1 balance)
Xtr, Xte, ytr, yte = train_test_split(
    X, y, test_size=0.25, random_state=SEED, stratify=y
)

# 3) Train model
model = RandomForestClassifier(n_estimators=200, random_state=SEED)
model.fit(Xtr, ytr)

# 4) Evaluate (labels specified so reports never crash even if a class is rare)
pred = model.predict(Xte)
print("Accuracy:", round(accuracy_score(yte, pred), 3))

print("\nClassification Report:\n",
      classification_report(
          yte, pred,
          labels=[0, 1],
          target_names=["Reject", "Approve"],
          zero_division=0
      ))

print("\nConfusion Matrix:\n",
      confusion_matrix(yte, pred, labels=[0, 1]))

# 5) Simple predictor function
def approve_loan(income, credit_score, dti, years, loan_amount):
    x = np.array([[income, credit_score, dti, years, loan_amount]])
    p = model.predict_proba(x)[0, 1]
    return ("Approve" if p >= 0.5 else "Reject", round(float(p), 2))

# Example call
print("\nExample:", approve_loan(4000, 700, 0.25, 3, 10000))
```

```
Accuracy: 0.945
```

```
Classification Report:
               precision    recall  f1-score   support

      Reject       0.96      0.93      0.94       100
     Approve       0.93      0.96      0.95       100

    accuracy                           0.94       200
   macro avg       0.95      0.95      0.94       200
weighted avg       0.95      0.94      0.94       200


Confusion Matrix:
 [[93  7]
 [ 4 96]]

Example: ('Approve', 0.99)
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Ran
  warnings.warn(
```

```
Start coding or generate with AI.
```