import pandas as pd

# Step 1: Load Lending Club dataset from Downloads

data\_path = "/Users/gres1/Downloads/loan\_data.csv"

lc\_df = pd.read\_csv(data\_path)

print("Shape:", lc\_df.shape)

print(lc\_df.head())

print("Columns:", lc\_df.columns.tolist())

X = lc\_df.drop("not.fully.paid", axis=1)

y = lc\_df["not.fully.paid"]

# Step 2: Check for missing values

null\_counts = lc\_df.isnull().sum()

print("Missing values per column:\n", null\_counts[null\_counts > 0])

# Quick summary: how many total missing?

print("\nTotal missing values:", lc\_df.isnull().sum().sum())

# Rebuild feature/target split if needed

X = lc\_df.drop("not.fully.paid", axis=1)

y = lc\_df["not.fully.paid"]

# Identify columns

cat\_cols = X.select\_dtypes(include=["object"]).columns.tolist()

num\_cols = [c for c in X.columns if c not in cat\_cols]

print("Categorical:", cat\_cols)

print("Numeric:", len(num\_cols))

import matplotlib.pyplot as plt

# Step 3: Target distribution

target\_counts = lc\_df['not.fully.paid'].value\_counts(normalize=False)

target\_perc = lc\_df['not.fully.paid'].value\_counts(normalize=True) \* 100

print("Counts:\n", target\_counts)

print("\nPercentages:\n", target\_perc)

# Plot

plt.bar(target\_counts.index, target\_counts.values, tick\_label=["Paid", "Default"])

plt.title("Loan Repayment vs Default Distribution")

plt.ylabel("Number of Loans")

plt.show()

rom sklearn.preprocessing import OneHotEncoder

# version-safe OneHotEncoder

if "sparse\_output" in OneHotEncoder.\_\_init\_\_.\_\_code\_\_.co\_varnames:

ohe = OneHotEncoder(handle\_unknown="ignore", sparse\_output=False, drop="first")

else:

ohe = OneHotEncoder(handle\_unknown="ignore", sparse=False, drop="first")

# fit on categorical columns

X\_cat = pd.DataFrame(

ohe.fit\_transform(X[cat\_cols]),

columns=ohe.get\_feature\_names\_out(cat\_cols),

index=X.index

)

# combine numeric + categorical

X\_num = X[num\_cols].reset\_index(drop=True)

X\_encoded = pd.concat([X\_num, X\_cat.reset\_index(drop=True)], axis=1)

print("Encoded shape:", X\_encoded.shape)

from imblearn.over\_sampling import SMOTE

sm = SMOTE(random\_state=42)

X\_resampled, y\_resampled = sm.fit\_resample(X\_encoded, y)

print("Before balancing:\n", y.value\_counts())

print("\nAfter balancing:\n", y\_resampled.value\_counts())

import matplotlib.pyplot as plt

# Before balancing

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)

y.value\_counts().plot(kind="bar", color=["skyblue","salmon"])

plt.title("Original Target Distribution")

plt.xticks([0,1], ["Repaid (0)", "Default (1)"], rotation=0)

plt.ylabel("Count")

# After balancing

plt.subplot(1,2,2)

y\_resampled.value\_counts().plot(kind="bar", color=["skyblue","salmon"])

plt.title("Balanced Target Distribution (SMOTE)")

plt.xticks([0,1], ["Repaid (0)", "Default (1)"], rotation=0)

plt.ylabel("Count")

plt.tight\_layout()

plt.show()

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_resampled, y\_resampled, test\_size=0.2, random\_state=42, stratify=y\_resampled

)

print("Train shape:", X\_train.shape, "| Test shape:", X\_test.shape)

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import recall\_score, precision\_score, roc\_auc\_score

# Simple NN

clf = MLPClassifier(

hidden\_layer\_sizes=(16,), # small hidden layer

activation="relu",

solver="adam",

batch\_size=128,

max\_iter=20,

early\_stopping=True,

n\_iter\_no\_change=3,

random\_state=42

)

# Train

clf.fit(X\_train\_scaled, y\_train)

# Predictions

y\_prob = clf.predict\_proba(X\_test\_scaled)[:, 1]

y\_pred = (y\_prob >= 0.5).astype(int)

# Metrics

print("Sensitivity (Recall):", recall\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("ROC AUC:", roc\_auc\_score(y\_test, y\_prob))

from sklearn.metrics import roc\_auc\_score, roc\_curve

import matplotlib.pyplot as plt

import numpy as np

# AUC score

auc = roc\_auc\_score(y\_test, y\_prob)

print(f"ROC AUC: {auc:.4f}")

# ROC curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

plt.figure(figsize=(6,6))

plt.plot(fpr, tpr, label=f"AUC = {auc:.3f}")

plt.plot([0,1],[0,1],'--', color='gray')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate (Sensitivity)")

plt.title("ROC Curve — Lending Club Loan Default")

plt.legend()

plt.show()

# Best threshold using Youden's J

j\_scores = tpr - fpr

best\_idx = np.argmax(j\_scores)

best\_thr = thresholds[best\_idx]

print(f"Best threshold (Youden J): {best\_thr:.4f}")