#Importing libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sklearn

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

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

import xgboost

from xgboost import XGBClassifier

from imblearn.over\_sampling import SMOTE

from collections import Counter

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import warnings

warnings.filterwarnings('ignore')

#Loading the data set

df = pd.read\_csv(r"C:\Users\abhis\Downloads\loan\_detection.csv")

print(df.head())

print(df.shape)

#EDA

#Finding missing or duplicate values

print(df.info())

print(df.describe())

print(df.nunique())

print(df.isnull().sum())

print(df.duplicated().sum())

print(df.columns)

print(df['Loan\_Status\_label'])

print(df['Loan\_Status\_label'].value\_counts())

# Imbalance Data ratio and reprensentation

print(round(len(df[df['Loan\_Status\_label']==0])/len(df)\*100, 2))

print(round(len(df[df['Loan\_Status\_label']==1])/len(df)\*100, 2))

A screen shot of a computer code

Description automatically generated

# Analyze categorical variables

categorical\_columns = [col for col in df.columns if 'job\_' in col or 'education\_' in col or 'marital\_' in col]

for col in categorical\_columns:

    sns.countplot(x=col, data=df)

    plt.title(f'Distribution of {col}')

    plt.xticks(rotation=90)

    plt.show()

plt.pie(df['Loan\_Status\_label'].value\_counts(), autopct='%1.0f%%',labels=['Not Eligible','Eligible'],

            startangle=60,shadow=True,explode=[0,0.2])

plt.title('Imbalanced Data Visulaization')

plt.show()

A blue pie chart with a triangle and a few orange text

Description automatically generated with medium confidence

#Outlier or Anomalies

def remove\_outliers\_iqr(df, columns):

    for col in columns:

        Q1 = df[col].quantile(0.25)

        Q3 = df[col].quantile(0.75)

        IQR = Q3 - Q1  # Interquartile Range

        lower\_bound = Q1 - 1.5 \* IQR

        upper\_bound = Q3 + 1.5 \* IQR

        df = df[(df[col] >= lower\_bound) & (df[col] <= upper\_bound)]

    return df

numerical\_columns = ['age', 'campaign', 'pdays', 'previous']

df\_cleaned = remove\_outliers\_iqr(df, numerical\_columns)

print("Original dataset shape:", df.shape)

print("Cleaned dataset shape (after removing outliers):", df\_cleaned.shape)



# Pairplot to analyze relationships after cleaning

sns.pairplot(df\_cleaned[['age', 'campaign', 'previous', 'Loan\_Status\_label']], hue='Loan\_Status\_label')

plt.title('Pairplot after cleaning')

plt.show()

A graph of a graph of a graph

Description automatically generated with medium confidence

# Visualizing distribution of age and other numeric columns

numeric\_columns = ['age', 'campaign', 'pdays', 'previous']

for col in numeric\_columns:

    sns.histplot(df\_cleaned[col], bins=30, kde=True)

    plt.title(f'Distribution of {col}')

    plt.show()

A graph of a distribution of age

Description automatically generated

A graph of a distribution of campaign

Description automatically generated

#Split the dataset into features and target variable

X = df\_cleaned.drop(columns=['Loan\_Status\_label'])  # Features

y = df\_cleaned['Loan\_Status\_label']  # Target (Loan Approved or Not)

#Check class distribution before resampling

print("Class distribution before SMOTE:", Counter(y))

#Apply SMOTE to balance the data

smote = SMOTE(random\_state=42)

X\_resampled, y\_resampled = smote.fit\_resample(X, y)

#Check class distribution after resampling

print("Class distribution after SMOTE:", Counter(y\_resampled))



#Splitting your data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, y\_resampled, test\_size=0.2, random\_state=42)

print(X\_train.shape)

print(X\_test.shape)



#Feature scaling

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

#Model Building

#Initialize Logistic Regression, Random Forest, and Decision Tree models

log\_reg = LogisticRegression(max\_iter=1000, random\_state=42)

rf\_model = RandomForestClassifier(random\_state=42)

dt\_model = DecisionTreeClassifier(random\_state=42)

# Train Logistic Regression

log\_reg.fit(X\_train\_scaled, y\_train)

# Train Random Forest

rf\_model.fit(X\_train, y\_train)

# Train Decision Tree

dt\_model.fit(X\_train, y\_train)

# Model Evaluation

# Logistic Regression Evaluation

y\_pred\_log\_reg = log\_reg.predict(X\_test\_scaled)

log\_reg\_accuracy = accuracy\_score(y\_test, y\_pred\_log\_reg)

print(f'Logistic Regression Accuracy: {log\_reg\_accuracy:.2f}')

# Random Forest Evaluation

y\_pred\_rf = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, y\_pred\_rf)

print(f'Random Forest Accuracy: {rf\_accuracy:.2f}')

# Decision Tree Evaluation

y\_pred\_dt = dt\_model.predict(X\_test)

dt\_accuracy = accuracy\_score(y\_test, y\_pred\_dt)

print(f'Decision Tree Accuracy: {dt\_accuracy:.2f}')

A black background with white text

Description automatically generated

# Confusion Matrix and Classification Report for Random Forest

print("\nConfusion Matrix for Random Forest:")

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_rf)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix for Random Forest')

plt.show()

A blue squares with numbers and a number

Description automatically generated

print("\nClassification Report for Random Forest:")

print(classification\_report(y\_test, y\_pred\_rf))

A screenshot of a computer screen

Description automatically generated

# Hyperparameter Tuning using GridSearchCV for Random Forest

param\_grid = {

    'n\_estimators': [100, 200],

    'max\_depth': [10, 20, None],

    'min\_samples\_split': [2, 5]

}

grid\_search = GridSearchCV(rf\_model, param\_grid, cv=3, scoring='accuracy', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# Best Model from GridSearchCV

best\_rf\_model = grid\_search.best\_estimator\_

print(f'Best Parameters: {grid\_search.best\_params\_}')

# Evaluate the tuned model

y\_pred\_best\_rf = best\_rf\_model.predict(X\_test)

best\_rf\_accuracy = accuracy\_score(y\_test, y\_pred\_best\_rf)

print(f'Best Random Forest Accuracy (after tuning): {best\_rf\_accuracy:.2f}')



# Confusion Matrix and Heatmap (After Tuning)

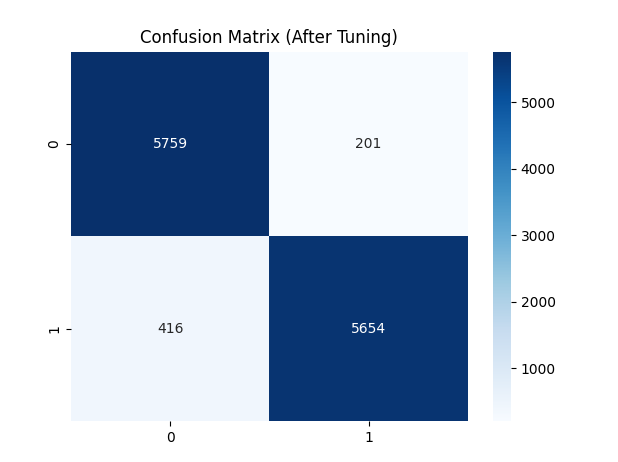
print("\nConfusion Matrix for Random Forest (After Tuning):")

conf\_matrix\_after = confusion\_matrix(y\_test, y\_pred\_best\_rf)

sns.heatmap(conf\_matrix\_after, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix (After Tuning)')

plt.show()



#Classification Report (Optional for After Tuning)

print("\nClassification Report for Random Forest (After Tuning):")

print(classification\_report(y\_test, y\_pred\_best\_rf))

A screenshot of a computer screen

Description automatically generated