**Comprehensive EDA and Data Preparation**

# **financial\_fraud\_eda.py**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

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

from imblearn.over\_sampling import SMOTE

import warnings

warnings.filterwarnings('ignore')

class FraudDetectionEDA:

def \_\_init\_\_(self, data\_path):

self.data = pd.read\_csv(data\_path)

self.preprocessed\_data = None

def initial\_inspection(self):

"""Perform initial data inspection"""

print("=== DATASET OVERVIEW ===")

print(f"Dataset shape: {self.data.shape}")

print("\nFirst 5 rows:")

print(self.data.head())

print("\n=== DATA TYPES ===")

print(self.data.dtypes)

print("\n=== MISSING VALUES ===")

missing\_data = self.data.isnull().sum()

print(missing\_data[missing\_data > 0])

print("\n=== BASIC STATISTICS ===")

print(self.data.describe())

def analyze\_class\_distribution(self):

"""Analyze the distribution of fraudulent vs legitimate transactions"""

fraud\_count = self.data['Class'].sum()

total\_count = len(self.data)

fraud\_percentage = (fraud\_count / total\_count) \* 100

print(f"\n=== CLASS DISTRIBUTION ===")

print(f"Legitimate transactions: {total\_count - fraud\_count} ({100-fraud\_percentage:.2f}%)")

print(f"Fraudulent transactions: {fraud\_count} ({fraud\_percentage:.4f}%)")

# **Visualization**

plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)

self.data['Class'].value\_counts().plot(kind='bar', color=['skyblue', 'coral'])

plt.title('Transaction Class Distribution')

plt.xlabel('Class (0: Legitimate, 1: Fraud)')

plt.ylabel('Count')

plt.xticks(rotation=0)

plt.subplot(1, 2, 2)

plt.pie([100-fraud\_percentage, fraud\_percentage],

labels=['Legitimate', 'Fraud'],

autopct='%1.3f%%',

colors=['lightgreen', 'lightcoral'])

plt.title('Class Distribution Percentage')

plt.tight\_layout()

plt.show()

def analyze\_feature\_distributions(self):

"""Analyze distributions of key features"""

# **Select a subset of features for visualization**

feature\_subset = ['Time', 'Amount', 'V1', 'V2', 'V3', 'V4']

plt.figure(figsize=(15, 10))

for i, feature in enumerate(feature\_subset, 1):

plt.subplot(2, 3, i)

# **Plot distribution for legitimate transactions**

legitimate\_data = self.data[self.data['Class'] == 0][feature]

fraud\_data = self.data[self.data['Class'] == 1][feature]

plt.hist(legitimate\_data, bins=50, alpha=0.7, label='Legitimate', color='blue', density=True)

plt.hist(fraud\_data, bins=50, alpha=0.7, label='Fraud', color='red', density=True)

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

plt.xlabel(feature)

plt.ylabel('Density')

plt.legend()

plt.tight\_layout()

plt.show()

def correlation\_analysis(self):

"""Perform correlation analysis"""

# **Compute correlation matrix**

correlation\_matrix = self.data.corr()

# **Plot correlation heatmap**

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, cmap='coolwarm', center=0,

square=True, linewidths=0.5)

plt.title('Feature Correlation Heatmap')

plt.tight\_layout()

plt.show()

# **Show correlations with the target variable**

target\_correlations = correlation\_matrix['Class'].sort\_values(ascending=False)

print("\n=== TOP 10 FEATURES CORRELATED WITH FRAUD ===")

print(target\_correlations.head(10))

print("\n=== BOTTOM 10 FEATURES CORRELATED WITH FRAUD ===")

print(target\_correlations.tail(10))

def handle\_class\_imbalance(self, method='smote'):

"""Handle class imbalance using various techniques"""

X = self.data.drop('Class', axis=1)

y = self.data['Class']

# **Split data before addressing imbalance**

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

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

)

print(f"Training set class distribution: {np.bincount(y\_train)}")

print(f"Test set class distribution: {np.bincount(y\_test)}")

if method == 'smote':

# **Apply SMOTE oversampling**

smote = SMOTE(random\_state=42)

X\_train\_balanced, y\_train\_balanced = smote.fit\_resample(X\_train, y\_train)

print(f"After SMOTE - Training set class distribution: {np.bincount(y\_train\_balanced)}")

elif method == 'undersampling':

# **Undersample majority class**

from imblearn.under\_sampling import RandomUnderSampler

undersampler = RandomUnderSampler(random\_state=42)

X\_train\_balanced, y\_train\_balanced = undersampler.fit\_resample(X\_train, y\_train)

print(f"After Undersampling - Training set class distribution: {np.bincount(y\_train\_balanced)}")

else:

# **Use class weights**

X\_train\_balanced, y\_train\_balanced = X\_train, y\_train

self.preprocessed\_data = {

'X\_train': X\_train\_balanced,

'y\_train': y\_train\_balanced,

'X\_test': X\_test,

'y\_test': y\_test

}

return self.preprocessed\_data

def scale\_features(self):

"""Scale numerical features"""

if self.preprocessed\_data is None:

raise ValueError("Please handle class imbalance first")

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(self.preprocessed\_data['X\_train'])

X\_test\_scaled = scaler.transform(self.preprocessed\_data['X\_test'])

self.preprocessed\_data['X\_train\_scaled'] = X\_train\_scaled

self.preprocessed\_data['X\_test\_scaled'] = X\_test\_scaled

self.preprocessed\_data['scaler'] = scaler

return self.preprocessed\_data

# **Main execution**

if \_\_name\_\_ == "\_\_main\_\_":

# Initialize EDA

eda = FraudDetectionEDA('creditcard.csv')

# **Perform comprehensive analysis**

eda.initial\_inspection()

eda.analyze\_class\_distribution()

eda.analyze\_feature\_distributions()

eda.correlation\_analysis()

# **Preprocess data**

preprocessed\_data = eda.handle\_class\_imbalance(method='smote')

scaled\_data = eda.scale\_features()

print("\n=== DATA PREPROCESSING COMPLETE ===")

print(f"Training set shape: {scaled\_data['X\_train\_scaled'].shape}")

print(f"Test set shape: {scaled\_data['X\_test\_scaled'].shape}")