**Fraud Prediction using Random Forest: End-to-End Process**

**1. Introduction**

Fraud detection is a critical task for financial institutions and businesses. This document outlines the end-to-end process of predicting fraudulent activities using a Random Forest model. The process includes data preparation, exploration, model training, and evaluation.

**2. Solution Architecture**

**2.1. Data Flow**

1. **Data Acquisition**: Import dataset from a CSV file.
2. **Data Preprocessing**: Clean and preprocess the data for analysis.
3. **Exploratory Data Analysis (EDA)**: Analyze the dataset to understand its characteristics and visualize important features.
4. **Feature Engineering**: Convert categorical variables into numerical format suitable for the Random Forest model.
5. **Model Training**: Split the data into training and testing sets and train the Random Forest model.
6. **Model Evaluation**: Assess the model's performance using various metrics.
7. **Results Visualization**: Generate visualizations to interpret the model's performance and insights.

**2.2. Components**

* **Data Handling Libraries**: Pandas, NumPy
* **Visualization Libraries**: Matplotlib, Seaborn
* **Machine Learning Libraries**: Scikit-learn (for preprocessing, model training, and evaluation)
* **Environment**: Python (Jupyter Notebook or any Python IDE)

**3. Methodology**

**3.1. Data Import and Preprocessing**

python

import pandas as pd

import numpy as np

# Load the dataset

data = pd.read\_csv("Fraud\_check.csv")

# Check the first and last few rows

print(data.head())

print(data.tail())

# Data Shape and Information

print(data.shape)

print(data.info())

# Check for missing and duplicate values

print(data.isnull().sum())

print(data.duplicated().sum())

**3.2. Exploratory Data Analysis (EDA)**

python

import matplotlib.pyplot as plt

import seaborn as sns

# Directory for saving figures

fig\_dir = "G:/TOP-MENTOR/FILES/Sep 1st/Assignments/Random Forests\_prj9 & 10/"

# Box Plot for numerical features

plt.figure(figsize=(8, 6))

sns.boxplot(data[['Taxable.Income', 'City.Population', 'Work.Experience']])

plt.title('Box Plot for Numerical Features')

plt.savefig(f"{fig\_dir}Box\_plot.jpg")

plt.show()

# Bar Plot for Taxable Income across Marital Status

plt.figure(figsize=(8, 6))

sns.barplot(data=data, x='Marital.Status', y='Taxable.Income', hue='Undergrad', dodge=True, errwidth=0)

plt.title("Taxable Income Across Marital Status")

plt.xlabel("Marital Status")

plt.ylabel("Taxable Income")

plt.savefig(f"{fig\_dir}Taxable\_Income\_across\_Marital\_Status.jpg")

plt.show()

# Kernel Density Plot

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)

sns.kdeplot(data=data, x='Taxable.Income', fill=True, color='blue')

plt.title('Taxable Income')

plt.subplot(1, 3, 2)

sns.kdeplot(data=data, x='City.Population', fill=True, color='red')

plt.title('City Population')

plt.subplot(1, 3, 3)

sns.kdeplot(data=data, x='Work.Experience', fill=True, color='orange')

plt.title('Work Experience')

plt.tight\_layout()

plt.savefig(f"{fig\_dir}KDE\_plot.jpg")

plt.show()

# Bar Plot for Taxable Income in Urban Areas

plt.figure(figsize=(8, 6))

sns.barplot(data=data, x='Urban', y='Taxable.Income', errwidth=0)

plt.title("Taxable Income in Urban Areas")

plt.xlabel("Urban")

plt.ylabel("Taxable Income")

plt.savefig(f"{fig\_dir}Taxable\_Income\_in\_Urban\_Areas.jpg")

plt.show()

**3.3. Data Preparation**

python

from sklearn.preprocessing import LabelEncoder

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

# Encode categorical variables

X = data[['Undergrad', 'Marital.Status', 'Work.Experience', 'Urban']]

X['Undergrad'] = X['Undergrad'].map({'NO': 0, 'YES': 1})

X['Urban'] = X['Urban'].map({'NO': 0, 'YES': 1})

X['Marital.Status'] = X['Marital.Status'].map({'Single': 0, 'Married': 1, 'Divorced': 2})

# Categorize the target variable

data['Taxable.Income'] = data['Taxable.Income'].apply(lambda x: 'Risky' if x <= 30000 else 'Good')

y = data['Taxable.Income']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=1234)

**3.4. Model Training and Evaluation**

python

from sklearn.ensemble import RandomForestClassifier

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

# Train the Random Forest model

rf = RandomForestClassifier(n\_estimators=500, max\_depth=4, random\_state=2, criterion='gini', min\_samples\_split=4, min\_samples\_leaf=3)

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

# Predict and evaluate

rf\_pred = rf.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, rf\_pred) \* 100

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

print("\nClassification Report:")

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

# Confusion Matrix

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

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cbar=False, cmap='viridis', xticklabels=['Good', 'Risky'], yticklabels=['Good', 'Risky'])

plt.title("Confusion Matrix")

plt.xlabel("Predicted Label")

plt.ylabel("Actual Label")

plt.show()

**4. Time Taken**

1. **Data Import and Preprocessing**: 1.0 sec
2. **Exploratory Data Analysis (EDA)**: 2.0 sec
3. **Data Preparation**: 1.5 sec
4. **Model Training and Evaluation**: 2.5 sec

**5. Conclusion**

This document provides a comprehensive overview of the process used for predicting fraudulent activities using a Random Forest model. The steps include data loading, preprocessing, exploratory analysis, model training, and evaluation, with relevant visualizations saved to the specified directory.