TASK 5

CREDIT CARD FRAUD DETECTION

- Build a machine learning model to identify fraudulent credit card transactions.
- Preprocess and normalize the transaction data, handle class imbalance issues, and split the dataset into training and testing sets.
- Train a classification algorithm, such as logistic regression or random forests, to classify transactions as fraudulent or genuine.
- Evaluate the model's performance using metrics like precision, recall, and F1-score, and consider techniques like oversampling or undersampling for improving results.

DATASET CLICK HERE

```python
import tkinter as tk
from tkinter import filedialog
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def select_file():
    """Opens a file dialog for user to select a file."""
    file_path = filedialog.askopenfilename()
    return file_path

def read_dataset(file_path):
    """Reads the dataset from the specified file path."""
    try:
        df = pd.read_csv(file_path)
        return df
    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
        return None
    except Exception as e:
        print(f"Error reading the file: {e}")
        return None

def preprocess_data(df):
    """Preprocesses the credit card fraud dataset."""
    # Handle missing values (if any)
    # df.fillna(method='ffill', inplace=True)  # Deprecated method
    df.ffill(inplace=True)  # Use ffill to forward-fill missing values

    # Separate features and target variable
    X = df.drop('Class', axis=1)
    y = df['Class']

    # Handle class imbalance (if necessary)
    # You can use techniques like oversampling, undersampling, or SMOTE
    # ...

    # Scale features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
    df.ffill(inplace=True)  # Use ffill to forward-fill missing values

    # Separate features and target variable
    X = df.drop('Class', axis=1)
    y = df['Class']

    # Handle class imbalance (if necessary)
    # You can use techniques like oversampling, undersampling, or SMOTE
    # ...

    # Scale features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test

def train_model(X_train, y_train, model_type='logistic_regression'):
    """Trains the specified model."""
    if model_type == 'logistic_regression':
        model = LogisticRegression()
    elif model_type == 'random_forest':
        model = RandomForestClassifier()
    else:
        raise ValueError("Invalid model type. Please choose 'logistic_regression' or 'random_forest'.")

    model.fit(X_train, y_train)
    return model

def evaluate_model(model, X_test, y_test):
    """Evaluates the model's performance."""
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)

def main():
    file_path = select_file()
```

```
    X = scaler.fit_transform(X)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test

def train_model(X_train, y_train, model_type='logistic_regression'):
    """Trains the specified model."""
    if model_type == 'logistic_regression':
        model = LogisticRegression()
    elif model_type == 'random_forest':
        model = RandomForestClassifier()
    else:
        raise ValueError("Invalid model type. Please choose 'logistic_regression' or 'random_forest'.")

    model.fit(X_train, y_train)
    return model

def evaluate_model(model, X_test, y_test):
    """Evaluates the model's performance."""
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)

def main():
    file_path = select_file()

    if file_path:
        df = read_dataset(file_path)

        if df is not None:
            X_train, X_test, y_train, y_test = preprocess_data(df)
            model = train_model(X_train, y_train, model_type='logistic_regression')  # You can change the model type here
            evaluate_model(model, X_test, y_test)

if __name__ == "__main__":
    main()
```