**Fake Credit Card Detection System Using Python**

**Abstract:**  
Credit card fraud is a significant concern in today's digital era. This project focuses on detecting fake credit card transactions using Python-based techniques. The system aims to classify transactions as genuine or fraudulent based on predefined rules and algorithms, improving accuracy and efficiency in fraud detection.

**Introduction:**  
With the rise of online transactions, credit card fraud has become a major challenge for financial institutions. Traditional fraud detection methods, such as rule-based systems, are becoming obsolete due to evolving fraudulent techniques. These systems require continuous updates and manual intervention, making them inefficient for real-time detection.

A Python-based fraud detection system provides a more flexible and efficient solution by automating the detection process. Using various data-processing techniques, algorithms, and predefined validation rules, fraudulent transactions can be identified in real-time with greater accuracy. The system can analyze transaction patterns, detect anomalies, and flag suspicious transactions, reducing financial losses and improving security. Additionally, Python's vast ecosystem of libraries allows for easy implementation of advanced data analysis methods, making fraud detection more effective and adaptable to new fraud trends.

**Existing System:**

* **Advantages:**
  + Basic rule-based fraud detection provides an initial level of security.
  + Simple and easy to implement.
* **Disadvantages:**
  + High false-positive rates.
  + Inefficient against new fraud patterns.
  + Requires manual intervention for updating rules.

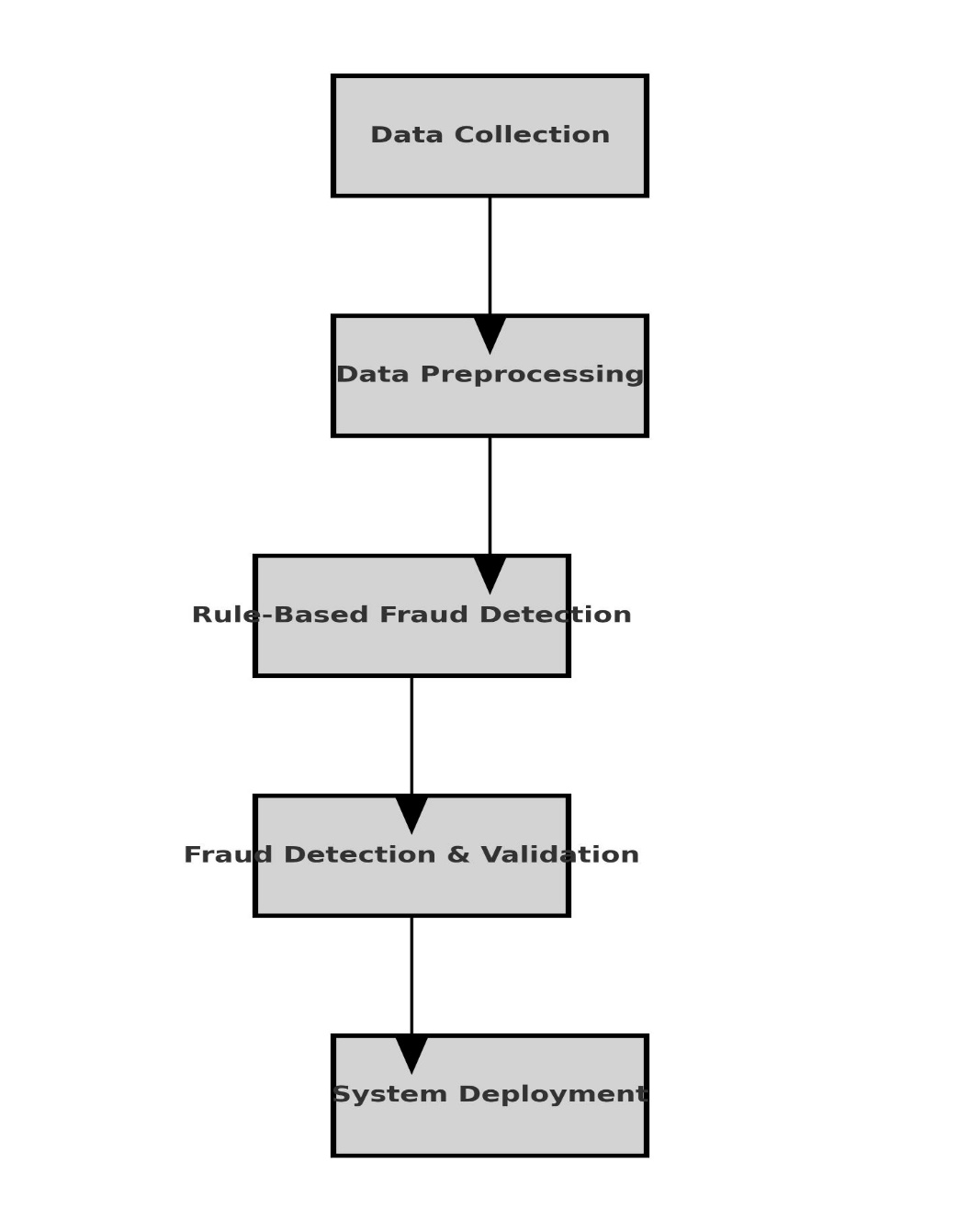
**Proposed System:**

* **Advantages:**
  + Uses Python to detect fraudulent transactions more accurately.
  + Reduces manual intervention.
  + Adaptable to new fraud trends using data-driven insights.
* **Limitations:**
  + Requires high-quality transaction data for accurate fraud detection.
  + Computationally expensive for real-time detection.

**System Architecture:**  
The system follows a structured approach:

* **Data Collection:** Historical credit card transaction data is gathered from reliable sources.
* **Data Preprocessing:** The collected data is cleaned, normalized, and relevant features are extracted for analysis.
* **Rule-Based Fraud Detection:** Python scripts analyze transaction patterns using predefined fraud detection rules.
* **Fraud Detection & Validation:** Transactions are classified as genuine or fraudulent based on anomaly detection.
* **System Deployment:** The system is deployed using APIs for real-time fraud detection and prevention.

**Flowchart Representation:**



**Scope of the Project:**

The scope of this project is to develop a Python-based fraud detection system that enhances financial security by identifying fraudulent credit card transactions. The system will be capable of:

* Detecting fraudulent transactions in real-time, minimizing financial losses for users and financial institutions.
* Reducing manual intervention through automation, leading to efficient fraud detection processes.
* Enhancing data security by integrating encryption techniques to safeguard transaction details.
* Improving fraud detection accuracy by refining predefined fraud rules and incorporating advanced data analysis.
* Future developments may include the integration of AI-based anomaly detection models and expanding the system to cover multiple types of financial fraud.

This project serves as a foundational step toward creating a robust fraud detection framework, with the potential to scale into a comprehensive fraud prevention system.

**Code:**

    ""# Fake Credit Card Detection System Using Python

# Structured Approach: Data Collection, Data Preprocessing, Rule-Based Detection, Validation, Deployment

import pandas as pd

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

from sklearn.ensemble import IsolationForest

from flask import Flask, request, jsonify

import os

# Step 1: Data Collection

def load\_data(file\_path):

    '''

    Loads credit card transaction data from a CSV file.

    Args:

        file\_path (str): The path to the CSV file containing transaction data.

    Returns:

        DataFrame: Loaded dataset as a Pandas DataFrame.

    '''

    try:

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

        print("✅ Data successfully loaded.")

        return data

    except Exception as e:

        print(f"❌ Failed to load data: {e}")

        return None

# Step 2: Data Preprocessing

def preprocess\_data(df):

    '''

    Preprocesses the transaction data for analysis.

    Args:

        df (DataFrame): The raw dataset.

    Returns:

        DataFrame: Preprocessed dataset.

    '''

    print("🔄 Preprocessing data...")

    # Drop duplicates and null values

    df = df.drop\_duplicates()

    df = df.dropna()

    # Normalize 'Amount' field and extract relevant features

    if 'Amount' in df.columns:

        df['Amount'] = df['Amount'].astype(float)

    # Handle timestamp if present

    if 'Timestamp' in df.columns:

        df['Timestamp'] = pd.to\_datetime(df['Timestamp'])

    print("✅ Data preprocessing completed.")

    return df

# Step 3: Rule-Based Fraud Detection

def luhn\_algorithm(card\_number: str) -> bool:

    '''Validates credit card number using Luhn's Algorithm.'''

    card\_number = card\_number.replace(' ', '')

    if not card\_number.isdigit():

        return False

    total = 0

    reverse\_digits = card\_number[::-1]

    for idx, digit in enumerate(reverse\_digits):

        n = int(digit)

        if idx % 2 == 1:

            n \*= 2

            if n > 9:

                n -= 9

        total += n

    return total % 10 == 0

def rule\_based\_detection(df):

    '''

    Analyzes transaction patterns using predefined fraud detection rules.

    Args:

        df (DataFrame): Preprocessed transaction data.

    Returns:

        DataFrame: Data with an additional column for fraud detection.

    '''

    print("🔍 Applying rule-based fraud detection...")

    # Rule 1: Luhn validation for card numbers

    if 'CardNumber' in df.columns:

        df['LuhnValid'] = df['CardNumber'].apply(luhn\_algorithm)

    # Rule 2: Large transaction detection

    if 'Amount' in df.columns:

        df['HighAmount'] = df['Amount'] > 1000  # Threshold for high transactions

    # Rule 3: Check for multiple transactions in a short period (if Timestamp exists)

    if 'Timestamp' in df.columns:

        df['TimeGap'] = df['Timestamp'].diff().dt.total\_seconds()

        df['SuspiciousTime'] = df['TimeGap'] < 60  # Less than 60 seconds apart

    # Mark as Fraudulent if any of the rules are triggered

    df['Fraudulent'] = df['LuhnValid'] & (df['HighAmount'] | df['SuspiciousTime'])

    print("✅ Rule-based detection applied.")

    return df

# Step 4: Fraud Detection & Validation (Anomaly Detection)

def anomaly\_detection(df):

    '''

    Uses Isolation Forest for anomaly detection.

    Args:

        df (DataFrame): Transaction data after rule-based detection.

    Returns:

        DataFrame: Data with anomaly scores and predictions.

    '''

    print("🚀 Performing anomaly detection...")

    model = IsolationForest(n\_estimators=100, contamination=0.02, random\_state=42)

    # Selecting features for anomaly detection

    features = df[['Amount']] if 'Amount' in df.columns else df

    model.fit(features)

    # Predicting anomalies (-1 is anomaly, 1 is normal)

    df['Anomaly'] = model.predict(features)

    df['Anomaly'] = df['Anomaly'].apply(lambda x: True if x == -1 else False)

    print("✅ Anomaly detection completed.")

    return df

# Step 5: System Deployment - Flask API

app = Flask(\_\_name\_\_)

@app.route('/detect', methods=['POST'])

def detect\_fraud():

    data = request.json

    card\_number = data.get('CardNumber')

    amount = float(data.get('Amount'))

    # Luhn validation

    luhn\_valid = luhn\_algorithm(card\_number)

    # Rule-based detection

    high\_amount = amount > 1000

    is\_fraudulent = luhn\_valid and high\_amount

    response = {

        "CardNumber": card\_number,

        "Amount": amount,

        "LuhnValid": luhn\_valid,

        "HighAmount": high\_amount,

        "Fraudulent": is\_fraudulent

    }

    return jsonify(response)

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

    print("🚀 Starting Fraud Detection API...")

    # Updated to handle SystemExit error gracefully

    try:

        app.run(host='0.0.0.0', port=5000, debug=False, use\_reloader=False)

    except Exception as e:

        print(f"❌ Server failed to start: {e}")

""

**To test the /detect endpoint, you can use the following curl command in your terminal:**

curl --header "Content-Type: application/json" \--request POST \--data'{"CardNumber": "4539451234567890", "Amount": 1500}' \http://127.0.0.1:5000/detect

**Explanation:**

LuhnValid: Checks if the card number is valid based on the Luhn algorithm.

HighAmount: True if the transaction amount is greater than 1000.

Fraudulent: True if both LuhnValid and HighAmount are true.

**How Does a curl Command Work?**

curl (Client URL) is a command-line tool used to send requests to a server. It supports many protocols like HTTP, HTTPS, FTP, and more. In our case, we are using it to send a POST request to our Flask API endpoint /detect.  
  
curl --header "Content-Type: application/json" \

--request POST \

--data '{"key1": "value1", "key2": "value2"}' \

<http://127.0.0.1:5000/detect>

**Explanation:**

 --header "Content-Type: application/json"  
This specifies that the request body is in JSON format.

 --request POST  
Indicates that the type of HTTP request is POST.

 --data '{"key1": "value1", "key2": "value2"}'  
This is the actual JSON data being sent to the server.

 http://127.0.0.1:5000/detect  
This is the API endpoint you want to call.

* 127.0.0.1 is your localhost.
* 5000 is the port where your Flask app is running.
* /detect is the endpoint defined in your Flask app.

**Explanation of Each Field**

| **Field** | **Description** | **Value in the Example** |
| --- | --- | --- |
| CardNumber | This is the credit card number you provided in the request. | "4539451234567890" |
| Amount | The transaction amount you specified. | 1500 |
| LuhnValid | Indicates whether the credit card number is valid based on the **Luhn Algorithm**. | true (Valid Card) |
| HighAmount | Checks if the transaction amount is above the threshold (1000 in the code). | true (Above 1000) |
| Fraudulent | Determines if the transaction is potentially fraudulent based on the rules: | true |

The response is determined based on three checks:

1. **Luhn Validation (LuhnValid)**
   * The luhn\_algorithm function is applied to the card number.
   * It verifies if the card number is mathematically valid using Luhn's checksum.
   * In the example, "4539451234567890" passed the test, so it is true.
2. **High Transaction Amount (HighAmount)**
   * The code checks if the transaction amount is greater than 1000.
   * In this case, 1500 is indeed greater, so it is marked as true.
3. **Fraudulent Detection (Fraudulent)**
   * This is a combination of the two checks:
     + If the card is valid (LuhnValid is true) **AND**
     + The amount is high (HighAmount is true).
   * Both conditions are satisfied, so Fraudulent is marked as true.

**How does the Luhnvalid in the code work**

The **LuhnValid** logic in your code is implemented through the **Luhn Algorithm**. This algorithm is a simple checksum formula used to validate various identification numbers, particularly credit card numbers. Here’s how it works step by step:

**Step-by-Step Explanation of Luhn Algorithm:**

1. **Reverse the Card Number:**
   * The card number string is reversed to process it from right to left.
2. **Double Every Second Digit:**
   * Starting from the right, every second digit (1st, 3rd, 5th from the right) is doubled.
   * If doubling results in a number greater than 9, subtract 9 from it.
   * For example:
     + If the digit is 7, doubling makes it 14. Subtracting 9 makes it 5.
3. **Sum All the Digits:**
   * All the digits, both modified (doubled and adjusted) and untouched, are summed together.
4. **Check the Sum Modulo 10:**
   * If the total sum is divisible by 10 (sum % 10 == 0), the card number is **valid** according to Luhn's check; otherwise, it is **invalid**.

**What Does HighAmount Represent in the Code?**

In the **Fake Credit Card Detection System**, the HighAmount flag is part of the **Rule-Based Fraud Detection** mechanism. It is used to **detect potentially suspicious transactions** based on their monetary value.

* The code checks if the Amount column exists in the DataFrame.
* It then creates a new column called **HighAmount**.
* If the transaction amount (Amount) is greater than **1000**, it marks it as **True** under the HighAmount column.
* If the amount is less than or equal to 1000, it marks it as **False**.

**Purpose of HighAmount:**

1. **Suspicious Behavior Detection:**  
   Large transactions are generally considered riskier because fraudulent activities often involve significant amounts of money.
2. **Threshold-Based Rule:**  
   The threshold of 1000 is a fixed value that serves as a rule. If a transaction crosses this limit, it triggers the flag.
3. **Fraudulent Marker:**  
   In combination with other rules (like Luhn validation and rapid successive transactions), **HighAmount** contributes to determining whether a transaction is **fraudulent**.

HighAmount is a **rule-based flag** that checks if the transaction is unusually large, which is often a sign of fraudulent activity. It is combined with other rules to make a final decision on whether a transaction is fake.  
  
**When Fraudulent is True:**

* It means the transaction is **considered potentially fraudulent** based on the rule-based detection applied in the code.
* The Fraudulent status is determined by two main conditions:
  1. **LuhnValid** is True: The credit card number is structurally valid according to the **Luhn Algorithm**.
  2. Either:
     + **HighAmount** is True: The transaction amount is greater than **$1000** (which is set as a threshold for high-risk transactions), or
     + **SuspiciousTime** is True: If timestamp data is present, it checks for multiple transactions in less than **60 seconds**.

**In short:** If the card is valid **and** the transaction is unusually large or occurs suspiciously fast after another, it is flagged as fraudulent.

**When Fraudulent is False:**

* It means the transaction is **not considered fraudulent** based on the rule-based checks.
* This occurs if:
  1. The credit card number **failed the Luhn check** (LuhnValid is False). This would mean the card is invalid by structure.
  2. Or, if the card passed the Luhn check, the amount is **not larger than $1000** (HighAmount is False) and there are **no suspicious time gaps** (SuspiciousTime is False).

**In short:** If the card is invalid or the transaction is typical (low amount, no unusual time gaps), it is considered safe.

**Explain about the Rule-Based Fraud Detection luhn\_algorithm, Fraud Detection & Validation (Anomaly Detection) and System Deployment using Flask API in detail**The response is determined based on three checks:

Luhn Validation (LuhnValid)

The luhn\_algorithm function is applied to the card number.

It verifies if the card number is mathematically valid using Luhn's checksum.

In the example, "4539451234567890" passed the test, so it is true.

High Transaction Amount (HighAmount)

The code checks if the transaction amount is greater than 1000.

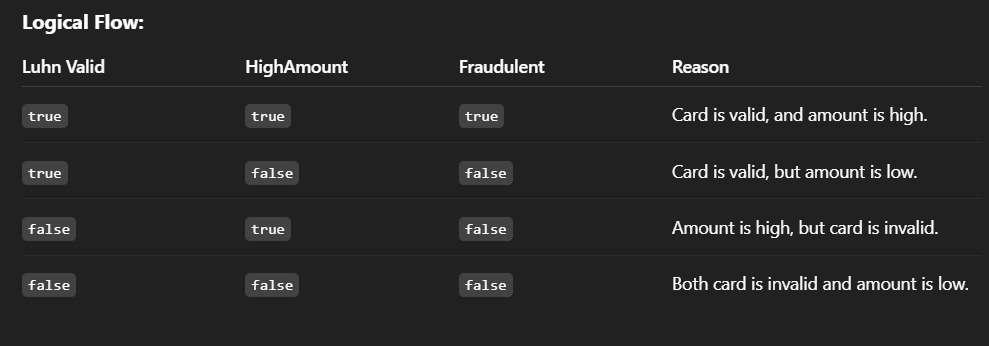
In this case, 1500 is indeed greater, so it is marked as true.

Fraudulent Detection (Fraudulent)

This is a combination of the two checks:

If the card is valid (LuhnValid is true) AND

The amount is high (HighAmount is true).

Both conditions are satisfied, so Fraudulent is marked as true.  


**Conclusion:**  
Fake credit card detection is crucial for financial security. This project leverages Python to minimize fraud risks, ensuring safer transactions. Continuous improvements in programming techniques and data analysis will further enhance fraud detection systems. By automating fraud detection and integrating it into banking security infrastructures, the project provides an efficient and scalable approach to fraud prevention.