## Detailed Code Walkthrough

The provided code is designed for monitoring blockchain transactions in real time, detecting anomalies (such as fake identities, Sybil attacks, and bot activities), and displaying results on an interactive dashboard. The application simulates transactions, processes them for anomalies, and provides a user interface with various graphs and alerts.

Let's break down the code into different sections to explain the functionality and how the code works:

---

## 1. Importing Libraries

The first part of the code imports several libraries for various functionalities:

```
import pandas as pd
import numpy as np
import random
import time
from datetime import datetime, timedelta
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, classification_report
import joblib
from web3 import Web3
import plotly.express as px
import plotly.graph_objects as go
from dash import Dash, dcc, html, Input, Output, State
import dash_bootstrap_components as dbc
import networkx as nx
import matplotlib.pyplot as plt
import socket
import signal
import sys
import logging
import os
import io
import base64
from PIL import Image
import google.generativeai as genai
```

- **Data Handling and Machine Learning**: Libraries like `pandas`, `numpy`, `sklearn`, and `joblib` are used for data processing, model training, and saving/loading models.
- **Blockchain**: `web3` is used to interact with the Ethereum blockchain.
- **Visualization**: `plotly`, `networkx`, and `matplotlib` are used to visualize data like transaction activity, user interaction, and wallet networks.
- **Web Framework**: `Dash`, along with `dash_bootstrap_components`, is used to create the interactive web dashboard.
- **Security & Blockchain**: `google.generativeai` for utilizing Google's Gemini model, `socket`, `signal`, and `sys` for managing server connections and graceful shutdowns.

---

## 2. Blockchain Setup

The code configures the blockchain connection and ledger to interact with Ethereum-like networks using Infura:

```
BLOCKCHAIN_API_KEY = "e65ee1b207274346b6a586c24e43bb18"
BLOCKCHAIN_URL = f"https://mainnet.infura.io/v3/{BLOCKCHAIN_API_KEY}"
w3 = Web3(Web3.HTTPProvider(BLOCKCHAIN_URL))
```

- **API Key**: The Infura API key is used to connect to the Ethereum mainnet via the Web3 provider.
- **BlockchainLedger Class**: This class stores verified addresses and defines methods to check if an address is valid or verified against a predefined list.

```
class BlockchainLedger:
    def __init__(self):
        self.verified_addresses = set([ ... ])  # List of pre-defined verified addresses
        self.w3 = w3

    def is_connected(self):
        return True  # Always returns True for simplicity in this setup

    def verify_address(self, address):
        return Web3.is_address(address) and address in self.verified_addresses

    def is_verified(self, address):
        return self.verify_address(address)
```

---

## 3. Transaction Simulation and Alerts Detection

The core functionality involves simulating real-time transactions and detecting potential anomalies:

**Simulating Real-Time Transactions**

```
def simulate_real_time_transaction():
    address = random.choice([ ... ])  # Choose a random address
    transaction = {
        'Sent tnx': random.randint(1, 10),  # Random transaction count
        'Received Tnx': random.randint(1, 10),
        'avg val sent': random.uniform(0.01, 5.0),  # Random amount
        'avg val received': random.uniform(0.01, 5.0),
        'Unique Sent To Addresses': random.randint(1, 20),  # Random number of addresses
        'Unique Received From Addresses': random.randint(1, 20),
        'Time Diff between first and last (Mins)': random.uniform(1, 120),  # Random time
        'Timestamp': datetime.now(),  # Current timestamp
        'Account Age': random.randint(1, 100),  # Random account age
        'Device Fingerprint': random.choice(['DeviceA', 'DeviceB', 'DeviceC']),
        'IP Address': random.choice(['192.168.0.1', '192.168.0.2']),
        'Geolocation': random.choice(['US', 'EU', 'Asia']),
        'Address': address
    }
    transaction_history.append(transaction)
    return transaction
```

This function generates a random transaction with various attributes (e.g., amount sent, address, timestamp, etc.). It simulates real-time activity on the blockchain.

**Detecting Alerts**

```
def detect_alerts(transaction_data, previous_data=None):
    global alerts, blocked_accounts
    current_alerts = []
    address = transaction_data['Address']

    # Skip if account is already blocked
    if address in blocked_accounts:
        current_alerts.append(f"Account {address} is already blocked.")
        alerts.extend(current_alerts)
        return

    df = pd.DataFrame(transaction_history)

    # Check for Sybil attacks based on IP or device fingerprint
    if len(df) > 1:
        similar_ip = df[df['IP Address'] == transaction_data['IP Address']]
        similar_device = df[df['Device Fingerprint'] == transaction_data['Device Fingerprint']]
        if len(similar_ip) > 3 or len(similar_device) > 3:
            current_alerts.append(f"Sybil Attack detected - Account {address} has been
automatically blocked.")
            block_account(address)

    # Analyze transaction data for fake identities, Sybil attacks, or bot activities
    features_fake = feature_engineering_real_time_fake(transaction_data, previous_data)
    features_sybil = feature_engineering_real_time_sybil(transaction_data, previous_data)
    features_bot = feature_engineering_real_time_bot(transaction_data, previous_data)

    # Predict anomalies based on trained models
    if pipeline_fake.predict(features_fake)[0] == 1:
        current_alerts.append(f"Fake Identity detected for account {address}.")
    if pipeline_sybil.predict(features_sybil)[0] == 1:
        current_alerts.append(f"Sybil Attack detected - Account {address} has been
automatically blocked.")
        block_account(address)
    if pipeline_bot.predict(features_bot)[0] == 1:
        current_alerts.append(f"Bot Activity detected - Account {address} has been
automatically blocked.")
        block_account(address)

    # Check if the address is verified in the blockchain ledger
    if not ledger.is_verified(address):
        current_alerts.append(f"Unverified Address: Blockchain check failed for account
{address}. Follow-up action recommended.")

    if current_alerts:
        alerts.extend(current_alerts)
```

This function performs the core task of alert detection based on transaction features and machine learning models (e.g., `pipeline_fake`, `pipeline_sybil`, and `pipeline_bot`). It checks for anomalies such as **Sybil attacks**, **fake identities**, and **bot activities**, and it automatically blocks suspicious accounts.

---

## 4. Graph-based Anomaly Detection

The blockchain transaction network is represented as a **directed graph**. The edges represent transactions, and the nodes represent wallets.

### Building the Graph

```
def build_wallet_graph(transactions):
    G = nx.DiGraph()
    for src, dst, amount in transactions:
        if G.has_edge(src, dst):
            G[src][dst]['weight'] += amount  # Aggregate amounts between the same wallets
        else:
            G.add_edge(src, dst, weight=amount)
    return G
```

This function builds a **directed graph** of wallet transactions using the `networkx` library, with the weight of the edges representing the transaction amount.

**Detecting Anomalies Using Isolation Forest**

```
def detect_graph_anomalies(features, contamination=0.1):
    nodes = list(features.keys())
    data = np.array(list(features.values()))

    clf = IsolationForest(random_state=42, contamination=contamination)
    clf.fit(data)
    preds = clf.predict(data)

    anomalies = [nodes[i] for i, pred in enumerate(preds) if pred == -1]

    # Add anomalies to the global list and block them
    for addr in anomalies:
        if addr not in anomalies_detected:
            anomalies_detected.append(addr)
            alerts.append(f"Graph Anomaly detected for account {addr}. Automatically blocked.")
            block_account(addr)

    return anomalies
```

This function uses the **Isolation Forest** algorithm to detect anomalous wallets based on their features (e.g., in-degree, out-degree, transaction volume). The detected anomalies are added to a global list and automatically blocked.

---

## 5. Dash Dashboard and Visualization

The code creates a **Dash** application for displaying various visualizations (e.g., heatmaps, graphs, and time series) of blockchain transactions.

```
app = Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])
```

This creates a Dash app with **Bootstrap** for styling.

**Dashboard Layout**

The dashboard layout consists of multiple **cards**, each displaying different types of data:

- **Transaction Activity Heatmap**
- **User Interaction Network**
- **Transaction Frequency and Amount Over Time**
- **Wallet Graph Anomaly Detection**
- **AI Security Analysis**

Each component (e.g., graphs, alert lists, blocked accounts) is rendered and updated at regular intervals using **Dash callbacks**.

**Real-Time Update of the Dashboard**

```
@app.callback(
   [Output('heatmap', 'figure'),
    Output('network-graph', 'figure'),
    Output('time-series', 'figure'),
    Output('alerts-list', 'children'),
    Output('blocked-accounts', 'children'),
    Output('graph-visualization', 'src')],
   [Input('interval-component', 'n_intervals')]
)
```

This callback triggers the **real-time update** of the dashboard, where:

- Transactions are simulated every 5 seconds (`interval-component`).
- The alert detection logic is executed.
- Anomalies are detected using both **graph-based** and **model-based** methods.
- The data is then visualized and displayed on the dashboard.

---

## 6. Generating LLM Insights

Finally, the code provides **LLM-powered insights** based on detected alerts and anomalies.

```
def get_llm_insights(alerts_list, anomalies_list):
   try:
      prompt = "You are a blockchain security analyst. Analyze the following alerts and
detected anomalies..."
      response = model.generate_content(prompt)
      llm_insights = response.text
      return llm_insights
   except Exception as e:
      logger.error(f"Error getting LLM insights: {e}")
      return f"Error getting AI insights: {str(e)}"
```

This function sends the **alerts** and **anomalies** to Google's **Gemini model** (using the `google.generativeai` API) to get security insights and recommendations.

## Conclusion

The provided code is a **comprehensive blockchain security monitoring** system that:

- Simulates blockchain transactions in real time.
- Detects anomalies (fake identities, Sybil attacks, bot activities) using machine learning models and graph-based techniques.
- Displays visualizations and alerts through an interactive **Dash dashboard**.
- Leverages **Google Gemini** for AI-driven security insights.

The dashboard updates in real time, visualizing transaction activity, user interactions, and wallet network anomalies, while providing automated alerts and actions for suspicious behavior.

---

## Summarized Walkthrough: Blockchain Transaction Monitoring System

### 1. Initialization and Setup

- **Libraries**: The code uses libraries for data processing (`pandas`, `numpy`), machine learning (`sklearn`, `joblib`), blockchain interaction (`web3`), visualization (`plotly`, `networkx`), and web dashboard creation (`dash`, `dash_bootstrap_components`).
- **Blockchain Setup**:
  - The system connects to the Ethereum blockchain via the **Infura API**.
  - The `BlockchainLedger` class is responsible for verifying wallet addresses.

### 2. Simulating Real-Time Blockchain Transactions

- **Transaction Simulation**: The system simulates blockchain transactions by randomly generating:
  - **Transaction details**: Amount sent/received, transaction frequency, IP address, geolocation, device fingerprint, and other transaction metadata.
  - **Random wallet addresses**: Both real (verified) and random addresses are used in the simulation.
- These transactions are stored in a **global list** (`transaction_history`) for further analysis.

### 3. Alert Detection and Anomaly Handling

- **Detecting Anomalies**:

    - **Fake Identity**: Identified when certain transaction behaviors (e.g., low transaction amounts, high frequency) match patterns of known fake identities.
    - **Sybil Attacks**: Detected if there are multiple transactions originating from the same IP or device fingerprint within a short time frame.
    - **Bot Activity**: Identified if transactions show consistent patterns like repetitive actions or excessive amounts in a short period.
- **Model-based Predictions**: Using pre-trained **machine learning models** (`pipeline_fake`, `pipeline_sybil`, `pipeline_bot`), predictions are made on the features extracted from each transaction.

- **Blocking Suspicious Accounts**: If a wallet is found to be associated with fake identities, Sybil attacks, or bot activities, the account is automatically blocked.

- **Unverified Addresses**: Accounts that do not exist in the predefined verified list are flagged as unverified, but not automatically blocked.

### 4. Graph-Based Anomaly Detection

- **Transaction Network Graph**:

    - The transactions are represented as a **directed graph** where nodes represent wallets, and edges represent transactions.
    - Features like **in-degree**, **out-degree**, and **transaction volumes** are calculated for each node.
- **Isolation Forest for Anomaly Detection**: Using the **Isolation Forest** algorithm, the graph features are analyzed to detect anomalous wallet behavior, such as outlier transaction volumes or abnormal wallet interactions.

- **Blocking Anomalous Wallets**: Suspicious wallets identified as anomalies in the graph are also flagged and automatically blocked.

### 5. Real-Time Dashboard and Visualization

- The **Dash dashboard** is set up to display various visualizations and monitor the real-time transaction activity.

### Dashboard Layout and Components

1. **Alerts**: Displays alerts about suspicious accounts, including fake identities, Sybil attacks, and bot activity.
2. **Blocked Accounts**: Shows a list of accounts that have been blocked due to suspicious behavior.
3. **Transaction Activity Heatmap**: Shows a heatmap of transaction activity by **geolocation**, indicating where most transactions are occurring.
4. **User Interaction Network**: A network graph showing how wallets (users) interact with each other. Suspicious or blocked accounts are highlighted in **red**.

5. **Time Series**: Displays **transaction frequency** and **average transaction amounts** over time.
6. **Wallet Graph Visualization**: A graph visualization of wallet transactions, with anomalous wallets highlighted.
7. **AI Security Analysis**: Provides insights from **Google Gemini** (an LLM model), analyzing the alerts and anomalies detected.

## 6. How the Dashboard Works

The dashboard is updated every 5 seconds to reflect new transaction data, with the following key actions:

- **Simulate New Transactions**: Every 5 seconds, a new transaction is simulated.
- **Detect Alerts**: The system checks if the transaction exhibits any suspicious behavior (fake identity, Sybil attack, bot activity).
- **Run Graph-Based Anomaly Detection**: Every 5th iteration, a graph of wallet transactions is built and analyzed for anomalies.
- **Generate Visualizations**: Visual components (heatmap, network graph, time series, and wallet graph) are updated to reflect the latest data.

### Results Displayed on the Dashboard

1. **Transaction Activity Heatmap**:

    - **Purpose**: Shows transaction volumes across different geolocations (e.g., US, EU, Asia).
    - **Result**: A **heatmap** that visually indicates which regions are most active in terms of transaction volume, with color intensity representing the volume of transactions.
2. **User Interaction Network**:

    - **Purpose**: Displays the relationships between wallets based on transactions.
    - **Result**: A **network graph** where:
        - **Blue nodes** represent normal wallets.
        - **Red nodes** represent blocked (anomalous) wallets.
        - **Edges** represent transaction flows, with line thickness indicating transaction amounts.
3. **Time Series Chart**:

    - **Purpose**: Displays trends in transaction frequency and average transaction amounts over time.
    - **Result**: A **line graph** with:
        - One line showing **transaction frequency** (number of transactions).
        - Another line showing the **average transaction amount** over time.
4. **Wallet Graph Visualization**:

    - **Purpose**: Visualize wallet interactions and highlight anomalies.

    - **Result**: A **directed graph** where anomalous wallets (detected through graph-based anomaly detection) are highlighted in **red**.
5. **AI Security Analysis**:
    - **Purpose**: Summarizes insights from Google's **Gemini** model based on detected anomalies.
    - **Result**: A **text summary** that:
        - Provides recommendations and analysis based on the alerts and anomalies detected.
        - Helps identify potential issues or patterns in transaction data.

**How the Results Are Presented**

- **Alerts**:
    - The system continuously generates alerts based on the behavior of wallets. Each alert is associated with a specific **type of anomaly**, such as:
        - Fake Identity Detection: Alerting when a wallet is suspected of being a fake identity.
        - Sybil Attack: Triggered when a wallet is part of a network of accounts trying to manipulate the system.
        - Bot Activity: Alerting when transaction patterns match bot-like behavior (e.g., automated transactions).
- These alerts are displayed as a list on the dashboard.

- **Blocked Accounts**:
    - Suspicious accounts identified by the system are added to a **blocked accounts list**. This list is updated in real-time and displayed on the dashboard.
- **Graph Anomalies**:
    - Anomalies detected via graph analysis are highlighted on the **wallet graph** visualization, providing insights into which wallets are engaging in suspicious behaviors, such as interacting with multiple other wallets in an unusual pattern.

**Intermediate Steps and Results**

- **Transaction Data Simulation**: Random transactions are simulated and stored in a global list (`transaction_history`).
- **Alert Detection**: The system checks each transaction for potential anomalies and updates alerts accordingly.
- **Graph-based Anomaly Detection**: The wallet interaction graph is analyzed periodically for anomalous patterns using **Isolation Forest**.
- **Real-Time Updates**: The dashboard updates every 5 seconds to reflect the latest simulated transaction data and detected anomalies.

**Detailed Results (Bullet Points)**

- **Transaction Activity Heatmap**:
  - Displays transaction activity across different regions (e.g., US, EU, Asia).
  - Shows areas with high transaction volumes, helping to identify regions with unusual activity.
- **User Interaction Network**:
  - Shows the connections between wallets.
  - **Red nodes** indicate blocked or anomalous wallets, while **blue nodes** represent normal wallets.
  - Helps identify suspicious interactions between wallets.
- **Time Series**:
  - Tracks transaction frequency and transaction amounts over time.
  - Helps visualize trends, spikes, or drops in activity that may indicate fraudulent behavior or anomalies.
- **Wallet Graph Visualization**:
  - Visualizes wallet interactions as a directed graph.
  - **Red nodes** highlight wallets that have been flagged as suspicious or anomalous.
  - **Edges** represent transactions, with the weight of the edge representing the amount.
- **AI Security Analysis**:
  - Provides insights from Google's **Gemini model** based on the detected alerts and anomalies.
  - Summarizes the detected issues, providing recommendations for further action.

---

## Conclusion

The system continuously monitors blockchain transactions for signs of suspicious behavior using both **rule-based logic** (for Sybil attacks, fake identities, and bot activity) and **machine learning models**. It then provides a comprehensive view of the blockchain network's activity through the **Dash dashboard**, visualizing transaction patterns, user interactions, and detected anomalies. The system’s ability to automatically block suspicious accounts ensures proactive security measures.