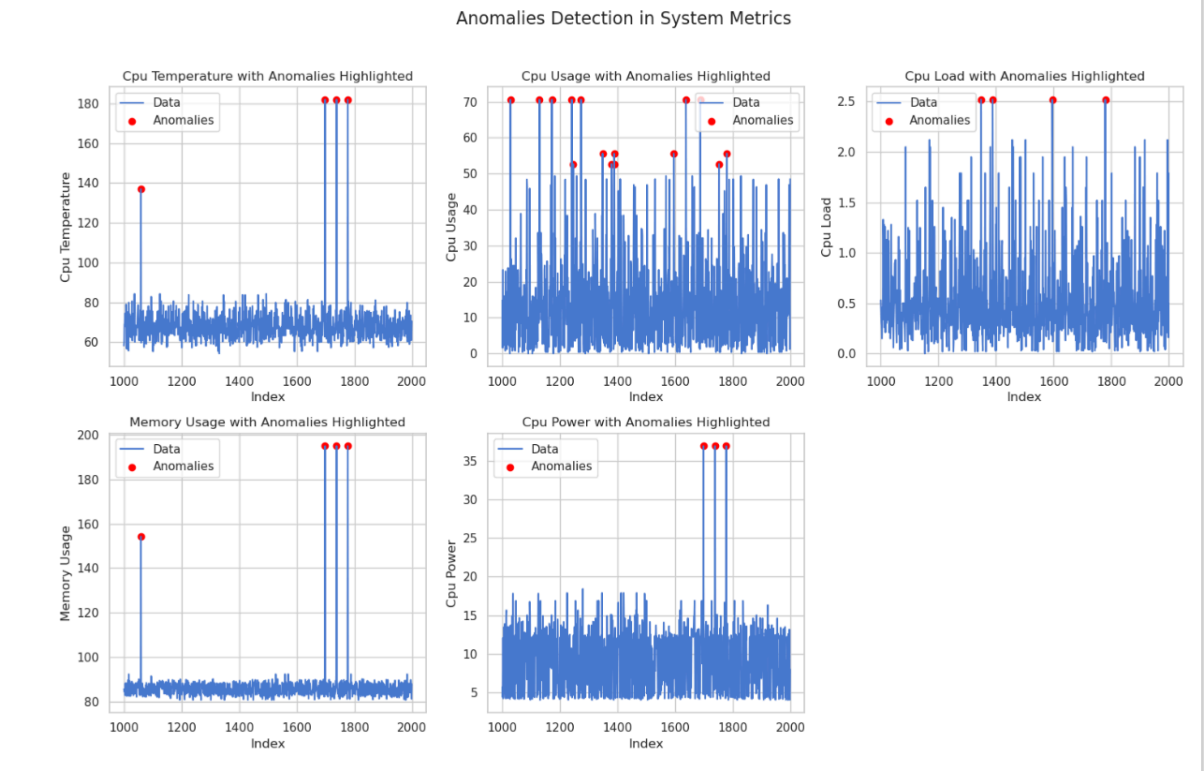
Create a synthetic multivariate time series data of at least 1 GB using the following code. And apply any simple anomaly detection algorithms and display the results like shown below



```
import wmi
import psutil
import pandas as pd
import datetime
import time
import random

# Settings
duration_minutes = 120
sampling_rate_hz = 10
num_samples = duration_minutes * 60 * sampling_rate_hz

# Start time
start_time = datetime.datetime.now()

# Initialize WMI client
w = wmi.WMI(namespace="root\\OpenHardwareMonitor")
```

```
# Initialize lists to store data
timestamps = []
cpu_temperatures = []
cpu_usages = []
cpu_loads = []
memory_usages = []
battery_levels = []
cpu_powers = []

# Collect data
for i in range(num_samples):
   try:
       # Get current time
       current_time = datetime.datetime.now()
       timestamps.append(current_time)

       # Get CPU temperature
       sensor_info = w.Sensor()
       cpu_temp = None
       cpu_power = None

       for sensor in sensor_info:
           if sensor.SensorType == 'Temperature' and 'CPU' in
sensor.Name:
               cpu_temp = sensor.Value
           if sensor.SensorType == 'Power' and 'CPU Package' in
sensor.Name:
               cpu_power = sensor.Value

       cpu_temperatures.append(cpu_temp)
       cpu_powers.append(cpu_power)

       # Get CPU usage
       cpu_usage = psutil.cpu_percent(interval=1/sampling_rate_hz)
       cpu_usages.append(cpu_usage)

       # Get CPU load (1 minute average)
       cpu_load = psutil.getloadavg()[0]
       cpu_loads.append(cpu_load)

       # Get memory usage
       memory_usage = psutil.virtual_memory().percent
       memory_usages.append(memory_usage)
```

```
        # Get battery level
        battery = psutil.sensors_battery()
        battery_level = battery.percent if battery else None
        battery_levels.append(battery_level)

        # Introduce anomalies randomly (e.g., 10% chance)
        if random.random() < 0.1:  # 10% chance to introduce anomaly
            # Introduce high CPU usage
            cpu_usages[-1] = random.uniform(90, 100)  # High CPU usage
        if random.random() < 0.1:
            # Introduce high temperature
            cpu_temperatures[-1] = random.uniform(90, 105)  # High
temperature
        if random.random() < 0.1:
            # Introduce high memory usage
            memory_usages[-1] = random.uniform(95, 100)  # High memory
usage
        if random.random() < 0.1:
            # Introduce low battery level
            battery_levels[-1] = random.uniform(0, 10)  # Low battery
level
        if random.random() < 0.1:
            # Introduce high CPU power
            cpu_powers[-1] = random.uniform(50, 100)  # Unusually high
CPU power

    except Exception as e:
        print(f"Error collecting data: {e}")
        cpu_temperatures.append(None)
        cpu_usages.append(None)
        cpu_loads.append(None)
        memory_usages.append(None)
        battery_levels.append(None)
        cpu_powers.append(None)

    # Create DataFrame
    data = {
        'timestamp': timestamps,
        'cpu_temperature': cpu_temperatures,
        'cpu_usage': cpu_usages,
        'cpu_load': cpu_loads,
        'memory_usage': memory_usages,
        'battery_level': battery_levels,
        'cpu_power': cpu_powers
    }

    df_real = pd.DataFrame(data)

    # Save to CSV in append mode

df_real.to_csv(r'C:\Users\Admin\Downloads\TSADSD-main\hardware_monitor_data.csv', mode='a', index=False)

    # Wait for the next sample
    time.sleep(1 / sampling_rate_hz)

# Display the first few rows of the DataFrame
df_real.head()
```