```
"""
異常検知用サンプルデータ生成プログラム（CSV形式）
ネットワークトラフィック、システムログ、センサーデータを生成
"""

import csv
import random
from datetime import datetime, timedelta
from typing import List, Dict
import os


class SampleDataGenerator:
    """サンプルデータ生成クラス"""

    def __init__(self):
        self.output_dir = "sample_data"
        os.makedirs(self.output_dir, exist_ok=True)

    def generate_network_traffic(self, num_normal: int = 10, num_anomaly: int = 5) -> List[Dict]:
        """ネットワークトラフィックデータ生成"""

        base_time = datetime.now() - timedelta(hours=24)

        # 正常なトラフィックパターン
        normal_patterns = [
            {
                "dst_ips": ["8.8.8.8", "8.8.4.4", "1.1.1.1"],  # DNS
                "ports": [443, 53],
                "protocols": ["HTTPS", "DNS"],
                "packets_range": (50, 300),
                "bytes_range": (10000, 100000)
            },
            {
                "dst_ips": ["10.0.0.5", "10.0.0.10", "192.168.10.5"],  # 社内サーバー
                "ports": [22, 80, 443, 3306],
                "protocols": ["SSH", "HTTP", "HTTPS", "MySQL"],
                "packets_range": (30, 200),
                "bytes_range": (5000, 80000)
            },
            {
                "dst_ips": ["172.217.175.46", "142.250.196.78"],  # Google services
                "ports": [443, 80],
                "protocols": ["HTTPS", "HTTP"],
                "packets_range": (100, 500),
                "bytes_range": (20000, 150000)
            }
        ]

        # 異常なトラフィックパターン
        anomaly_patterns = [
            {
                "name": "Torネットワーク通信",
                "dst_ips": ["185.220.101.5", "185.220.101.10", "45.61.185.90"],
                "ports": [9050, 9051, 9001],
                "protocols": ["TCP", "Tor"],
                "packets_range": (3000, 10000),
                "bytes_range": (1000000, 5000000)
            },
            {
                "name": "ポートスキャン",
                "dst_ips": ["複数の外部IP"],
                "ports": [445, 139, 135, 3389],
                "protocols": ["SMB", "NetBIOS", "RDP"],
                "packets_range": (5000, 20000),
                "bytes_range": (200000, 1000000)
            },
            {
                "name": "DDoS攻撃",
                "dst_ips": ["203.0.113.10"],
                "ports": [80, 443],
                "protocols": ["HTTP", "HTTPS"],
                "packets_range": (50000, 100000),
                "bytes_range": (5000000, 20000000)
            },
            {
                "name": "データ窃取",
                "dst_ips": ["91.203.5.165", "185.82.217.99"],
                "ports": [443, 8080],
                "protocols": ["HTTPS", "HTTP"],
                "packets_range": (10000, 30000),
                "bytes_range": (10000000, 50000000)
            },
            {
                "name": "C&Cサーバー通信",
                "dst_ips": ["45.142.212.61"],
                "ports": [4444, 8443],
                "protocols": ["TCP", "HTTPS"],
```

```
        "packets_range": (100, 500),
        "bytes_range": (5000, 50000)
    }
]

traffic_data = []

# 正常トラフィック生成
for i in range(num_normal):
    pattern = random.choice(normal_patterns)
    src_ip = f"192.168.1.{random.randint(10, 100)}"
    timestamp = base_time + timedelta(minutes=random.randint(0, 1440))

    traffic = {
        "id": i + 1,
        "timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "src_ip": src_ip,
        "dst_ip": random.choice(pattern["dst_ips"]),
        "port": random.choice(pattern["ports"]),
        "protocol": random.choice(pattern["protocols"]),
        "packets": random.randint(*pattern["packets_range"]),
        "bytes": random.randint(*pattern["bytes_range"]),
        "label": "normal"
    }
    traffic_data.append(traffic)

    # 異常トラフィック生成
    for i in range(num_anomaly):
        pattern = random.choice(anomaly_patterns)
        src_ip = f"192.168.1.{random.randint(10, 100)}"
        timestamp = base_time + timedelta(minutes=random.randint(0, 1440))

        traffic = {
            "id": num_normal + i + 1,
            "timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "src_ip": src_ip,
            "dst_ip": random.choice(pattern["dst_ips"]),
            "port": random.choice(pattern["ports"]),
            "protocol": random.choice(pattern["protocols"]),
            "packets": random.randint(*pattern["packets_range"]),
            "bytes": random.randint(*pattern["bytes_range"]),
            "label": "anomaly"
        }
        traffic_data.append(traffic)

    # 時系列順にソート
    traffic_data.sort(key=lambda x: x["timestamp"])

    # IDを振り直し
    for i, traffic in enumerate(traffic_data, 1):
        traffic["id"] = i

    return traffic_data

def generate_system_logs(self, num_normal: int = 15, num_anomaly: int = 8) -> List[Dict]:
    """システムログデータ生成"""

    base_time = datetime.now() - timedelta(hours=24)

    # 正常なログパターン
    normal_log_templates = [
        ("INFO", "User '{user}' logged in from {ip}"),
        ("INFO", "Database backup completed successfully"),
        ("INFO", "Service '{service}' started successfully"),
        ("INFO", "Cron job '{job}' executed successfully"),
        ("INFO", "User '{user}' logged out"),
        ("INFO", "Configuration file reloaded"),
        ("WARNING", "Disk usage at 75% on /dev/sda1"),
        ("INFO", "SSL certificate renewed successfully"),
        ("INFO", "Email sent to {email}"),
        ("INFO", "API request from {ip} completed in 250ms")
]

# 異常なログパターン
anomaly_log_templates = [
    ("WARNING", "Failed login attempt for user '{user}' from {suspicious_ip} (attempt {attempt})"),
    ("ERROR", "SQL injection attempt detected in parameter '{param}': {payload}"),
    ("CRITICAL", "Unauthorized file access attempt: {file} by process '{process}'"),
    ("ERROR", "Buffer overflow detected in application '{app}'"),
    ("CRITICAL", "Privilege escalation attempt by user '{user}'"),
    ("ERROR", "Cross-site scripting (XSS) attempt in input field '{field}'"),
    ("WARNING", "Suspicious process '{process}' connecting to external IP {ip}"),
    ("CRITICAL", "Ransomware behavior detected: Mass file encryption in {directory}"),
    ("ERROR", "Directory traversal attempt: {path}"),
    ("CRITICAL", "Root access from unauthorized IP: {ip}")
]

users = ["admin", "user01", "user02", "developer", "operator"]
```

```
services = ["apache2", "mysql", "nginx", "docker"]
jobs = ["backup_db", "log_rotation", "cleanup_temp"]
suspicious_ips = ["185.220.101.5", "45.142.212.61", "91.203.5.165", "103.253.145.12"]
internal_ips = ["192.168.1.10", "192.168.1.25", "10.0.0.15"]

logs = []

# 正常ログ生成
for i in range(num_normal):
    level, template = random.choice(normal_log_templates)
    timestamp = base_time + timedelta(minutes=random.randint(0, 1440))

    message = template.format(
        user=random.choice(users),
        ip=random.choice(internal_ips),
        service=random.choice(services),
        job=random.choice(jobs),
        email=f"{random.choice(users)}@example.com"
    )

    logs.append({
        "id": i + 1,
        "timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "level": level,
        "message": message,
        "label": "normal"
    })

# 異常ログ生成
for i in range(num_anomaly):
    level, template = random.choice(anomaly_log_templates)
    timestamp = base_time + timedelta(minutes=random.randint(0, 1440))

    message = template.format(
        user=random.choice(users),
        suspicious_ip=random.choice(suspicious_ips),
        attempt=random.randint(5, 20),
        param=random.choice(["user_id", "username", "email", "search"]),
        payload="' OR '1'='1",
        file=random.choice(["/etc/passwd", "/etc/shadow", "/root/.ssh/id_rsa"]),
        process=random.choice(["unknown_proc", "malware.exe", "suspicious_script.sh"]),
        app=random.choice(["web_app", "api_service", "backend"]),
        field=random.choice(["comment", "description", "name"]),
        ip=random.choice(suspicious_ips),
        directory=random.choice(["/home/user/documents", "/var/www/html", "/opt/data"]),
        path="../../../etc/passwd"
    )

    logs.append({
        "id": num_normal + i + 1,
        "timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "level": level,
        "message": message,
        "label": "anomaly"
    })

# 時系列順にソート
logs.sort(key=lambda x: x["timestamp"])

# IDを振り直し
for i, log in enumerate(logs, 1):
    log["id"] = i

    return logs

def generate_sensor_data(self, duration_minutes: int = 60, interval_minutes: int = 5) -> List[Dict]:
    """センサー時系列データ生成"""

    base_time = datetime.now() - timedelta(minutes=duration_minutes)
    sensor_data = []

    # 正常動作の基準値
    normal_temp = 22.0
    normal_vibration = 0.05

    # 異常発生タイミング（全体の60%経過後）
    anomaly_start = int((duration_minutes / interval_minutes) * 0.6)

    data_points = duration_minutes // interval_minutes

    for i in range(data_points):
        timestamp = base_time + timedelta(minutes=i * interval_minutes)

        if i < anomaly_start:
            # 正常範囲内のランダムな変動
            temperature = normal_temp + random.uniform(-0.5, 0.5)
            vibration = normal_vibration + random.uniform(-0.01, 0.01)
            label = "normal"
```

```
    else:
        # 異常発生：時間経過とともに悪化
        progress = (i - anomaly_start) / (data_points - anomaly_start)

        # 温度上昇（指数関数的）
        temp_increase = (progress ** 2) * 30
        temperature = normal_temp + temp_increase + random.uniform(-1, 1)

        # 振動増加（線形 + ランダムスパイク）
        vib_increase = progress * 5
        if random.random() < 0.3:  # 30%の確率でスパイク
            vib_increase += random.uniform(0.5, 2.0)
        vibration = normal_vibration + vib_increase + random.uniform(-0.1, 0.1)

        label = "anomaly"

    sensor_data.append({
        "id": i + 1,
        "timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "temperature": round(temperature, 2),
        "vibration": round(vibration, 3),
        "label": label
    })

return sensor_data
```

```
def save_to_csv(self, data: List[Dict], filename: str):
    """CSVファイルに保存"""
    if not data:
        print(f"✗ データが空です: {filename}")
        return

    filepath = os.path.join(self.output_dir, filename)
    keys = data[0].keys()

    with open(filepath, 'w', newline='', encoding='utf-8') as f:
        writer = csv.DictWriter(f, fieldnames=keys)
        writer.writeheader()
        writer.writerows(data)

    print(f"✓ CSVファイル保存: {filepath}")

def generate_all_samples(self,
                         traffic_normal=100, traffic_anomaly=20,
                         log_normal=150, log_anomaly=30,
                         sensor_duration=300, sensor_interval=5):
    """全サンプルデータを生成して保存

    Args:
        traffic_normal: ネットワークトラフィックの正常データ数
        traffic_anomaly: ネットワークトラフィックの異常データ数
        log_normal: システムログの正常データ数
        log_anomaly: システムログの異常データ数
        sensor_duration: センサーデータの記録時間（分）
        sensor_interval: センサーデータの記録間隔（分）
    """
    print("=" * 70)
    print("サンプルデータ生成プログラム（CSV形式）")
    print("=" * 70)
    print()

    # 1. ネットワークトラフィックデータ
    print("【1. ネットワークトラフィックデータ生成】")
    traffic_data = self.generate_network_traffic(
        num_normal=traffic_normal,
        num_anomaly=traffic_anomaly
    )
    self.save_to_csv(traffic_data, "network_traffic.csv")
    print(f"   生成件数: 正常={traffic_normal}, 異常={traffic_anomaly}, 合計={len(traffic_data)}")
    print()

    # 2. システムログデータ
    print("【2. システムログデータ生成】")
    log_data = self.generate_system_logs(
        num_normal=log_normal,
        num_anomaly=log_anomaly
    )
    self.save_to_csv(log_data, "system_logs.csv")
    print(f"   生成件数: 正常={log_normal}, 異常={log_anomaly}, 合計={len(log_data)}")
    print()

    # 3. センサーデータ
    print("【3. センサー時系列データ生成】")
    sensor_data = self.generate_sensor_data(
        duration_minutes=sensor_duration,
        interval_minutes=sensor_interval
    )
```

```
self.save_to_csv(sensor_data, "sensor_data.csv")
sensor_points = sensor_duration // sensor_interval
print(f"   生成件数: {len(sensor_data)}データポイント ({sensor_duration}分間, {sensor_interval}分間隔)")
print()

print("=" * 70)
print("✓ 全サンプルデータの生成が完了しました")
print("=" * 70)
print()
print("【生成されたファイル】")
print(f"  • {os.path.join(self.output_dir, 'network_traffic.csv')}")
print(f"  • {os.path.join(self.output_dir, 'system_logs.csv')}")
print(f"  • {os.path.join(self.output_dir, 'sensor_data.csv')}")
print()
print("【CSVファイル形式】")
print("  ネットワークトラフィック:")
print("    id, timestamp, src_ip, dst_ip, port, protocol, packets, bytes, label")
print("  システムログ:")
print("    id, timestamp, level, message, label")
print("  センサーデータ:")
print("    id, timestamp, temperature, vibration, label")
print()
print("【使用方法】")
print("  これらのCSVファイルを異常検知プログラムで読み込んで使用できます")
print("  python anomaly_detection.py")
print()


def display_sample_preview(data: List[Dict], title: str, num_items: int = 3):
    """サンプルデータのプレビュー表示"""
    print(f"¥n【{title} - プレビュー】")
    print("-" * 70)

    if not data:
        print("  データがありません")
        return

    # ヘッダー表示
    headers = list(data[0].keys())
    print("  " + " | ".join(headers))
    print("  " + "-" * 60)

    # データ表示
    for item in data[:num_items]:
        values = [str(item[key]) for key in headers]
        print("  " + " | ".join(values))

    print("-" * 70)


def main():
    """メイン実行"""
    import argparse

    parser = argparse.ArgumentParser(
        description='異常検知用サンプルデータ生成プログラム（CSV形式）',
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
使用例:
  # デフォルト設定で生成（正常100件、異常20件など）
  python generate_sample_data.py

  # カスタム設定で生成
  python generate_sample_data.py --traffic-normal 200 --traffic-anomaly 50

  # センサーデータを24時間分生成（1分間隔）
  python generate_sample_data.py --sensor-duration 1440 --sensor-interval 1

  # すべてカスタマイズ
  python generate_sample_data.py ¥¥
    --traffic-normal 500 --traffic-anomaly 100 ¥¥
    --log-normal 1000 --log-anomaly 200 ¥¥
    --sensor-duration 720 --sensor-interval 5
    """
    )

    # ネットワークトラフィック設定
    parser.add_argument('--traffic-normal', type=int, default=100,
                        help='ネットワークトラフィックの正常データ数（デフォルト: 100）')
    parser.add_argument('--traffic-anomaly', type=int, default=20,
                        help='ネットワークトラフィックの異常データ数（デフォルト: 20）')

    # システムログ設定
    parser.add_argument('--log-normal', type=int, default=150,
                        help='システムログの正常データ数（デフォルト: 150）')
    parser.add_argument('--log-anomaly', type=int, default=30,
                        help='システムログの異常データ数（デフォルト: 30）')
```

```
# センサーデータ設定
parser.add_argument('--sensor-duration', type=int, default=300,
                    help='センサーデータの記録時間（分）（デフォルト: 300）')
parser.add_argument('--sensor-interval', type=int, default=5,
                    help='センサーデータの記録間隔（分）（デフォルト: 5）')

# プレビュー表示オプション
parser.add_argument('--no-preview', action='store_true',
                    help='プレビュー表示をスキップ')

args = parser.parse_args()

# バリデーション
if args.traffic_normal < 1 or args.traffic_anomaly < 1:
    print("エラー: トラフィックデータ数は1以上である必要があります")
    return

if args.log_normal < 1 or args.log_anomaly < 1:
    print("エラー: ログデータ数は1以上である必要があります")
    return

if args.sensor_duration < args.sensor_interval:
    print("エラー: センサー記録時間は記録間隔以上である必要があります")
    return

# データ生成
generator = SampleDataGenerator()

print(f"¥n設定:")
print(f"  ネットワークトラフィック: 正常={args.traffic_normal}, 異常={args.traffic_anomaly}")
print(f"  システムログ: 正常={args.log_normal}, 異常={args.log_anomaly}")
print(f"  センサーデータ: {args.sensor_duration}分間 / {args.sensor_interval}分間隔")
print()

# サンプルデータ生成
generator.generate_all_samples(
    traffic_normal=args.traffic_normal,
    traffic_anomaly=args.traffic_anomaly,
    log_normal=args.log_normal,
    log_anomaly=args.log_anomaly,
    sensor_duration=args.sensor_duration,
    sensor_interval=args.sensor_interval
)

# プレビュー表示
if not args.no_preview:
    print("¥n" + "=" * 70)
    print("サンプルデータプレビュー（最初の3件）")
    print("=" * 70)

    # ネットワークトラフィック
    traffic = generator.generate_network_traffic(num_normal=2, num_anomaly=1)
    display_sample_preview(traffic, "ネットワークトラフィック")

    # システムログ
    logs = generator.generate_system_logs(num_normal=2, num_anomaly=1)
    display_sample_preview(logs, "システムログ")

    # センサーデータ
    sensors = generator.generate_sensor_data(duration_minutes=15, interval_minutes=5)
    display_sample_preview(sensors, "センサーデータ", num_items=3)

    print("¥n" + "=" * 70)
    print("完了: 異常検知プログラムで使用できます")
    print("=" * 70)
    print("¥n次のステップ:")
    print("  python anomaly_detection.py")


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