Info Needed from Person 1(PCAP & Feature Extraction)

- List of all fields / keys the extractor will produce
- Data types for each field
- Optional vs required fields
- Sample feature data (mock dictionary or small JSON file)

Info Needed from Person 2 (Dataset Builder)

- Column names and order for the final CSV
- Labeling convention (how “Normal” vs “Abnormal” is assigned)
- 1–3 sample CSV rows (mock/fake data is fine)

PERSON 3: Validation & Schema Designer

Step 1: Dataset Schema

Basic field definition to build off

Attribute Name	Description	Synonym(s)	Data Type	Size (*=max)	Possible Data Values	Optional	Validation Rules
timestamp	Time when the packet/session occurred	time, datetime	datetime64	*	Any valid timestamp	No	Must be a valid datetime; cannot be null
source_ip	Source IP address of the packet/session	src_ip	str	45	Valid IPv4 or IPv6 address	No	Must match IP format (IPv4/IPv6); cannot be null
destination_ip	Destination IP address	dst_ip	str	45	Valid IPv4 or IPv6 address	No	Must match IP format; cannot be null
source_port	Source port number	src_port	int	5	0–65535	No	Must be integer in range 0–65535
destination_port	Destination port number	dst_port	int	5	0–65535	No	Must be integer in range 0–65535
protocol	Transport protocol	proto	str	10	TCP, UDP, ICMP	No	Must be one of allowed protocols
packet_count	Number of packets in the session	pkt_count	int	*	≥ 0	No	Must be non-negative integer
byte_count	Total bytes transferred in the session	bytes, total_bytes	int	*	≥ 0	No	Must be non-negative integer

session_duration	Duration of the session in seconds	duration, time_span	float	*	≥ 0	No	Must be non-negative float
label	Traffic classification (normal/abnormal)	class, category	str	20	Normal, Abnormal, [other traffic types]	No	Must match allowed values
domain_name	Domain name involved in the session (if any)	host	str	255	Any valid domain name	Yes	Optional; if present, must be valid domain format
min_packet_size	Minimum packet size in the session	pkt_min	int	*	≥ 0	Yes	Must be non-negative integer; optional if unavailable
max_packet_size	Maximum packet size in the session	pkt_max	int	*	≥ 0	Yes	Must be non-negative integer; optional if unavailable
avg_packet_size	Average packet size in the session	pkt_avg	float	*	≥ 0	Yes	Must be non-negative float; optional if unavailable

Step 2: Create Validation Rules (Point Form)

A. Identify Validation Checks

- **Required Columns**
- **Data Types**
- **Numeric Ranges**
- **Categorical Values**
- **Format Checks**
- **Anomaly Checks**
- **Missing Values**

B. Define Validation Rules for Schema.md

- **timestamp:** valid timestamp, not null
- **source_ip / destination_ip:** valid IPv4/IPv6, not null
- **source_port / destination_port:** integer 0–65535, not null
- **protocol:** one of [TCP, UDP, ICMP], not null
- **packet_count / byte_count / session_duration:** ≥ 0, not null
- **label:** one of allowed categories, not null
- **min/max/avg_packet_size:** optional; if present, ≥ 0 and min ≤ max

- **domain_name:** optional; if present, valid domain format

PYTHON VALIDATION CODE sample:
This stub covers:

- Required columns check
- Data types check
- Numeric range check
- Categorical values check
- IP and domain format check
- Reports invalid rows clearly

process_dataset.py:

```
import pandas as pd
 import numpy as np
 import re
------------------------------
Helper functions for validation
------------------------------
def is_valid_ip(ip):
 """Check if the IP address is a valid IPv4 or IPv6"""
 ipv4_pattern = r'^(\d{1,3}.){3}\d{1,3}$'
 ipv6_pattern = r'([0-9a-fA-F]{0,4}:){2,7}[0-9a-fA-F]{0,4}'
 return bool(re.match(ipv4_pattern, ip)) or bool(re.match(ipv6_pattern, ip))
def is_valid_domain(domain):
 """Check if domain name is valid"""
 domain_pattern = r'^(?:[a-zA-Z0-9-]+.)+[a-zA-Z]{2,}$'
 return bool(re.match(domain_pattern, domain))
------------------------------
Validator function
------------------------------
def validate_dataset(df: pd.DataFrame):
 errors = []
# Required columns
required_cols = [
    'timestamp', 'source_ip', 'destination_ip',
    'source_port', 'destination_port', 'protocol',
    'packet_count', 'byte_count', 'session_duration', 'label'
]

for col in required_cols:
    if col not in df.columns:
        errors.append(f"Missing required column: {col}")

if errors:
    return errors

# Data type checks
if not pd.api.types.is_datetime64_any_dtype(df['timestamp']):
    errors.append("timestamp column must be datetime")
for col in ['source_ip', 'destination_ip', 'protocol', 'label']:
    if not pd.api.types.is_string_dtype(df[col]):
        errors.append(f"{col} column must be string")
for col in ['source_port', 'destination_port', 'packet_count', 'byte_count']:
    if not pd.api.types.is_integer_dtype(df[col]):
        errors.append(f"{col} column must be integer")
```

```
if not pd.api.types.is_float_dtype(df['session_duration']):
    errors.append("session_duration column must be float")

# Value range checks
for col, min_val, max_val in [('source_port', 0, 65535), ('destination_port', 0, 65535),
                              ('packet_count', 0, None), ('byte_count', 0, None),
                              ('session_duration', 0, None)]:
    if max_val:
        invalid = df[(df[col] < min_val) | (df[col] > max_val)]
    else:
        invalid = df[df[col] < min_val]
    if not invalid.empty:
        errors.append(f"Invalid values in {col}: {len(invalid)} rows")

# Categorical value checks
allowed_protocols = ['TCP', 'UDP', 'ICMP']
allowed_labels = ['Normal', 'Abnormal']
invalid_protocols = df[~df['protocol'].isin(allowed_protocols)]
if not invalid_protocols.empty:
    errors.append(f"Invalid protocols: {len(invalid_protocols)} rows")
invalid_labels = df[~df['label'].isin(allowed_labels)]
if not invalid_labels.empty:
    errors.append(f"Invalid labels: {len(invalid_labels)} rows")

# IP and domain checks
for idx, row in df.iterrows():
    if not is_valid_ip(row['source_ip']):
        errors.append(f"Invalid source_ip at row {idx}")
    if not is_valid_ip(row['destination_ip']):
        errors.append(f"Invalid destination_ip at row {idx}")
    if 'domain_name' in df.columns and pd.notnull(row['domain_name']):
        if not is_valid_domain(row['domain_name']):
            errors.append(f"Invalid domain_name at row {idx}")

return errors

------------------------------
Example usage
------------------------------
if name == "main":
 # Load sample dataset
 df = pd.read_csv("sample_dataset.csv", parse_dates=['timestamp'])
# Validate dataset
validation_errors = validate_dataset(df)

if validation_errors:
    print("Validation errors found:")
    for err in validation_errors:
        print("-", err)
else:
    print("Dataset is valid!")
```

Step 3: Implement Validator Script

Implement helper functions

- is_valid_ip(ip) → check IPv4/IPv6
- is_valid_domain(domain) → check valid domain format

Check required columns

- Verify all columns defined in the schema are present
- Return an error list if missing

Validate data types

- Ensure each column has the correct type: int, float, str, datetime64

Validate numeric ranges

- Check source_port, destination_port in 0–65535
- Check packet_count, byte_count, session_duration ≥ 0

Validate categorical values

- protocol must be in [TCP, UDP, ICMP]
- label must be in allowed traffic types

Validate IPs and domains

- Check if source_ip and destination_ip are valid
- Check optional domain_name if present

Check anomalies

- min_packet_size > max_packet_size
- avg_packet_size < 0
- session_duration = 0 with packet_count > 0
- byte_count < packet_count

Test the script:

- Create a small CSV with valid and invalid rows
- Run the script and confirm it flags errors correctly

PYTHON VALIDATION CODE sample:

Features of sample Validator:

- Checks required columns, types, ranges, and categories
- Validates IP addresses and optional domain names
- Detects anomalies (impossible or suspicious data)
- Prints all errors with row numbers for easy debugging
- Ready to plug into your dataset pipeline

Process_dataset.py

```
import pandas as pd
 import numpy as np
 import re
------------------------------
Helper functions for validation
------------------------------
def is_valid_ip(ip):
 """Check if the IP address is a valid IPv4 or IPv6"""
 ipv4_pattern = r'^(\d{1,3}.){3}\d{1,3}$'
 ipv6_pattern = r'([0-9a-fA-F]{0,4}:){2,7}[0-9a-fA-F]{0,4}'
 return bool(re.match(ipv4_pattern, ip)) or bool(re.match(ipv6_pattern, ip))
```

```
def is_valid_domain(domain):
 """Check if domain name is valid"""
 domain_pattern = r'^(?:[a-zA-Z0-9-]+.)+[a-zA-Z]{2,}$'
 return bool(re.match(domain_pattern, domain))
```

Validator function

```
def validate_dataset(df: pd.DataFrame):
 errors = []
# Required columns
required_cols = [
    'timestamp', 'source_ip', 'destination_ip',
    'source_port', 'destination_port', 'protocol',
    'packet_count', 'byte_count', 'session_duration', 'label'
]

for col in required_cols:
    if col not in df.columns:
        errors.append(f"Missing required column: {col}")

if errors:
    return errors

# Data type checks
if not pd.api.types.is_datetime64_any_dtype(df['timestamp']):
    errors.append("timestamp column must be datetime")
for col in ['source_ip', 'destination_ip', 'protocol', 'label']:
    if not pd.api.types.is_string_dtype(df[col]):
        errors.append(f"{col} column must be string")
for col in ['source_port', 'destination_port', 'packet_count', 'byte_count']:
    if not pd.api.types.is_integer_dtype(df[col]):
        errors.append(f"{col} column must be integer")
if not pd.api.types.is_float_dtype(df['session_duration']):
    errors.append("session_duration column must be float")

# Value range checks
for col, min_val, max_val in [('source_port', 0, 65535), ('destination_port', 0, 65535),
                              ('packet_count', 0, None), ('byte_count', 0, None),
                              ('session_duration', 0, None)]:
    if max_val:
        invalid = df[(df[col] < min_val) | (df[col] > max_val)]
    else:
        invalid = df[df[col] < min_val]
    if not invalid.empty:
        errors.append(f"Invalid values in {col}: {len(invalid)} rows")

# Categorical value checks
allowed_protocols = ['TCP', 'UDP', 'ICMP']
allowed_labels = ['Normal', 'Abnormal']
invalid_protocols = df[~df['protocol'].isin(allowed_protocols)]
if not invalid_protocols.empty:
    errors.append(f"Invalid protocols: {len(invalid_protocols)} rows")
invalid_labels = df[~df['label'].isin(allowed_labels)]
if not invalid_labels.empty:
    errors.append(f"Invalid labels: {len(invalid_labels)} rows")
```

```
# IP and domain checks
for idx, row in df.iterrows():
    if not is_valid_ip(row['source_ip']):
        errors.append(f"Invalid source_ip at row {idx}")
    if not is_valid_ip(row['destination_ip']):
        errors.append(f"Invalid destination_ip at row {idx}")
    if 'domain_name' in df.columns and pd.notnull(row['domain_name']):
        if not is_valid_domain(row['domain_name']):
            errors.append(f"Invalid domain_name at row {idx}")

# Anomaly checks
anomaly_rows = df[
    (df.get('min_packet_size', 0) > df.get('max_packet_size', 0)) |
    (df.get('avg_packet_size', 0) < 0) |
    ((df['session_duration'] == 0) & (df['packet_count'] > 0)) |
    (df['byte_count'] < df['packet_count'])
]
if not anomaly_rows.empty:
    errors.append(f"Anomalies detected in {len(anomaly_rows)} rows")

return errors

------------------------------
Example usage
------------------------------
if name == "main":
 try:
 # Load sample dataset
 df = pd.read_csv("sample_dataset.csv", parse_dates=['timestamp'])
   # Validate dataset
    validation_errors = validate_dataset(df)

    if validation_errors:
        print("Validation errors found:")
        for err in validation_errors:
            print("-", err)
    else:
        print("Dataset is valid!")
except FileNotFoundError:
    print("Error: sample_dataset.csv not found.")
```

PERSON 4: Project Setup & Repo Lead

Repo structure:

```
network-traffic-dashboard/
│
├─ README.md                  # Project overview, installation, usage
├─ requirements.txt           # Python dependencies
├─ process_dataset.py         # Validator (Person 3)
├─ extract_features.py        # PCAP parsing
├─ build_dataset.py           # Dataset creation
├─ dashboard.py               # Streamlit dashboard
├─ sample_dataset.csv         # Example dataset
│
├─ tests/                     # Unit tests
│   ├─ test_validator.py
│   └─ __init__.py
│
├─ data/                      # Optional: PCAP files or CSVs
├─ docs/                      # Project documentation, schema.md
└─ .gitignore                 # Ignore Python cache, dataset files, etc.
```

BASIC UNIT TEST(Example test) in test_validator.py:

```
import pytest
import pandas as pd
from process_dataset import validate_dataset

def test_valid_dataset():
    df = pd.DataFrame({
        "timestamp": pd.to_datetime(["2025-11-20 10:00:00"]),
        "source_ip": ["192.168.1.1"],
        "destination_ip": ["192.168.1.2"],
        "source_port": [80],
        "destination_port": [443],
        "protocol": ["TCP"],
        "packet_count": [10],
        "byte_count": [500],
        "session_duration": [5.0],
        "label": ["Normal"]
    })
    errors = validate_dataset(df)
    assert errors == []
```

Run : pytest test/

Folder structure for tests

```
arduino

network-traffic-dashboard/
└─ tests/
    ├─ __init__.py
    ├─ test_validator.py
    └─ test_dashboard.py   # optional later
```

Additional Test to include in tests/test_validator.py:

✅ **What this covers:**

- Valid dataset passes
- Missing required columns
- Invalid IP detection
- Negative values and impossible ranges (anomalies)

Can expand tests for:

- extract_features.py (PCAP parsing)
- build_dataset.py (dataset creation logic)
- dashboard.py (Streamlit UI components)

```
import pytest
import pandas as pd
from process_dataset import validate_dataset

# ----------------------
# Test valid dataset
# ----------------------
def test_valid_dataset():
    df = pd.DataFrame({
        "timestamp": pd.to_datetime(["2025-11-20 10:00:00"]),
        "source_ip": ["192.168.1.1"],
        "destination_ip": ["192.168.1.2"],
        "source_port": [80],
        "destination_port": [443],
        "protocol": ["TCP"],
        "packet_count": [10],
        "byte_count": [500],
        "session_duration": [5.0],
        "label": ["Normal"]
    })
    errors = validate_dataset(df)
    assert errors == []

# ----------------------
# Test missing column
# ----------------------
def test_missing_column():
    df = pd.DataFrame({
        "timestamp": pd.to_datetime(["2025-11-20 10:00:00"]),
        "source_ip": ["192.168.1.1"]
        # missing destination_ip and others
    })
    errors = validate_dataset(df)
    assert any("Missing required column" in e for e in errors)

# ----------------------
# Test invalid IP
# ----------------------
def test_invalid_ip():
    df = pd.DataFrame({
        "timestamp": pd.to_datetime(["2025-11-20 10:00:00"]),
        "source_ip": ["999.999.999.999"],
        "destination_ip": ["192.168.1.2"],
        "source_port": [80],
        "destination_port": [443],
```

```
        "protocol": ["TCP"],
        "packet_count": [10],
        "byte_count": [500],
        "session_duration": [5.0],
        "label": ["Normal"]
    })
    errors = validate_dataset(df)
    assert any("Invalid source_ip" in e for e in errors)

# ----------------------
# Test negative values and anomalies
# ----------------------
def test_anomalies():
    df = pd.DataFrame({
        "timestamp": pd.to_datetime(["2025-11-20 10:00:00"]),
        "source_ip": ["192.168.1.1"],
        "destination_ip": ["192.168.1.2"],
        "source_port": [80],
        "destination_port": [443],
        "protocol": ["TCP"],
        "packet_count": [-5],
        "byte_count": [-100],
        "session_duration": [-1.0],
        "label": ["Normal"],
        "min_packet_size": [100],
        "max_packet_size": [50],
        "avg_packet_size": [-20]
    })
    errors = validate_dataset(df)
    assert any("Invalid values" in e or "Anomalies detected" in e for e in errors)
```