**INTELLIGENT SYSTEMS**

**CCINSYSL**

**COMPILATION OF ACTIVITIES**

|  |  |
| --- | --- |
| **ACT. NO.** | **TITLE** |
| 1 | Simulated Data Acquisition |
| 2 | Cleaning Raw Evidence |
| 3 | Feature Engineering |
| 4 | Automating the Triage Process |
| 5 | Introducing SpaCy: A Practical NLP Library |
| 6 | Introducing SpaCy: A Practical NLP Library 2 |
| 7 | Generating the Final Report |
| 8 | Detecting Suspicious Network Activity |
| 9 | Metadata Analysis and Source Identification |
| 10 | Advanced Report Generation and Final Project Kickoff |

Calub, John Paul S.

COM232

October 9, 2025

*Cover page*

Activity #: 1

Title: Simulated Data Acquisition

**Source Code**

# acquire\_data.py

import pandas as pd

import numpy as np

import random

from datetime import datetime, timedelta

# Create a rich, simulated dataset for our investigation

def generate\_raw\_evidence(num\_records=100):

    timestamps = [datetime.now() - timedelta(minutes=random.randint(1, 100)) for \_ in range(num\_records)]

    users = [f'user\_{random.randint(1, 10)}' for \_ in range(num\_records)]

    event\_types = ['login', 'logout', 'file\_access', 'network\_connection', 'email\_sent']

    events = [random.choice(event\_types) for \_ in range(num\_records)]

    # Generate a mix of messages, some with potential entities

    messages = []

    for i, event in enumerate(events):

        if event == 'login':

            messages.append(f'User logged in from IP 192.168.1.{random.randint(1, 254)}')

        elif event == 'network\_connection':

            messages.append(f'Connection to www.suspicious-site-{random.randint(1, 5)}.com on port 8080')

        elif event == 'file\_access':

            messages.append(f'Accessed sensitive document path: /home/{users[i]}/docs/private\_file\_{random.randint(1, 5)}.txt')

        elif event == 'email\_sent':

            messages.append('Email sent from Jane Doe to John Doe about Project X and the London office.')

        else:

            messages.append(f'{event} event.')

    # Simulate some missing data

    df = pd.DataFrame({

        'timestamp': timestamps,

        'user\_id': users,

        'event\_type': events,

        'message': messages

    })

    df.loc[df.sample(frac=0.1).index, 'user\_id'] = np.nan

    df.loc[df.sample(frac=0.05).index, 'message'] = None

    df.to\_csv('raw\_evidence.csv', index=False)

    print("Successfully generated 'raw\_evidence.csv' with simulated forensic data.")

if \_\_name\_\_ == '\_\_main\_\_':

    generate\_raw\_evidence()

Activity #: 1

Title: Simulated Data Acquisition

A close-up of a number

AI-generated content may be incorrect.A close-up of a number

AI-generated content may be incorrect.**Sample Output/Screen Shot**

A close-up of a document

AI-generated content may be incorrect.A close-up of a document

AI-generated content may be incorrect.

Activity #: 2

Title: Cleaning Raw Evidence

**Source Code**

# clean\_data.py

import pandas as pd

import os

raw\_data\_path = 'raw\_evidence.csv'

cleaned\_data\_path = 'cleaned\_evidence.csv'

if not os.path.exists(raw\_data\_path):

 print(f"Error: '{raw\_data\_path}' not found. Please complete Week 1 activity first.")

 exit()

df = pd.read\_csv(raw\_data\_path)

# Handle missing values

df.fillna('UNKNOWN', inplace=True)

# Convert timestamp column to datetime objects

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

# Save the cleaned data to a new CSV file

df.to\_csv(cleaned\_data\_path, index=False)

print(f"Successfully cleaned and saved data to '{cleaned\_data\_path}'.")

print("\nFirst 5 rows of cleaned data:")

print(df.head())

A close-up of a number

AI-generated content may be incorrect.Activity #: 2

Title: Cleaning Raw Evidence

**Sample Output/Screen Shot**

A close-up of a number

AI-generated content may be incorrect.

A close-up of a document

AI-generated content may be incorrect.A close-up of a document

AI-generated content may be incorrect.

Activity #: 3

Title: Feature Engineering

**Source Code**

import pandas as pd

def main():

    # Load the cleaned data

    df = pd.read\_csv("cleaned\_evidence.csv")

    # Ensure timestamp column exists

    if "timestamp" not in df.columns:

        raise ValueError("The file does not contain a 'timestamp' column.")

    # Convert timestamp to datetime

    df["timestamp"] = pd.to\_datetime(df["timestamp"], errors="coerce")

    # Feature engineering

    df["hour\_of\_day"] = df["timestamp"].dt.hour

    df["day\_of\_week"] = df["timestamp"].dt.day\_name()

    df["is\_weekend"] = df["day\_of\_week"].isin(["Saturday", "Sunday"])

    # Save enhanced data

    df.to\_csv("feature\_engineered\_evidence.csv", index=False)

    print("Feature engineering complete. File saved as feature\_engineered\_evidence.csv")

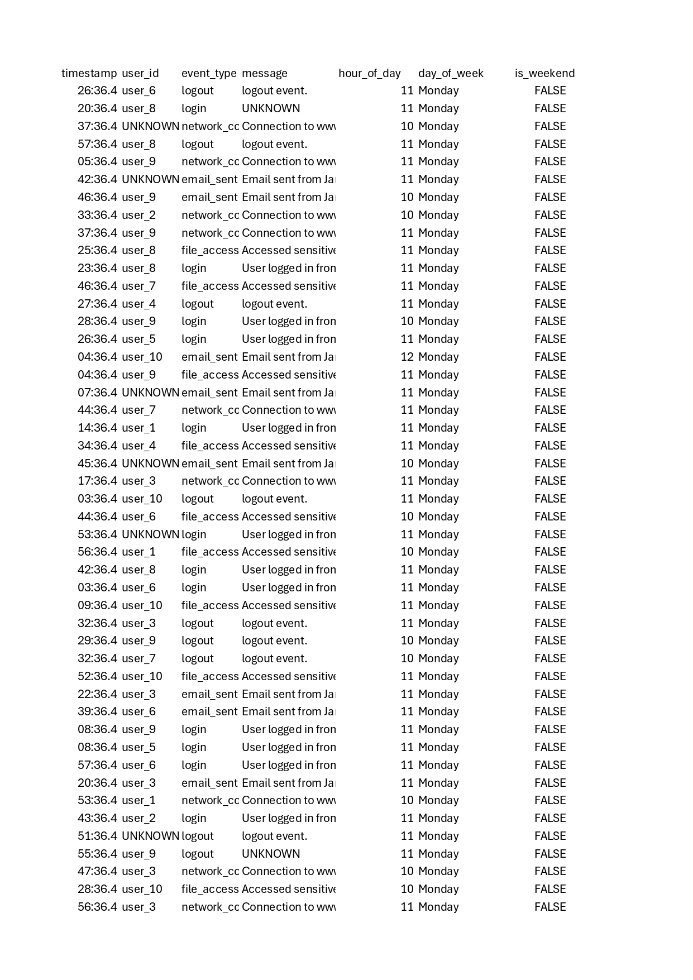
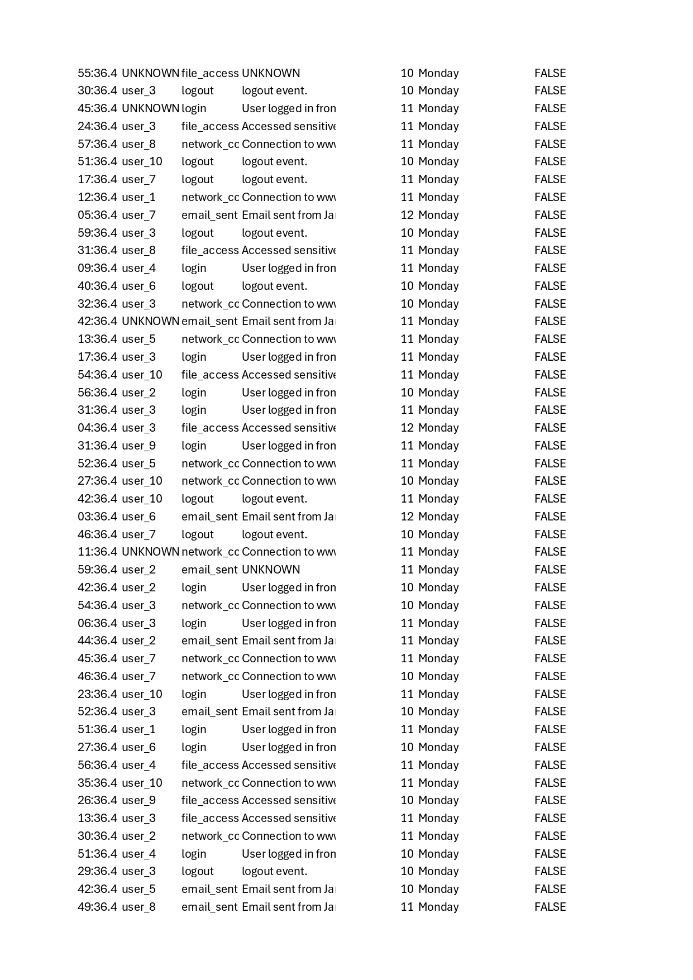
if \_\_name\_\_ == "\_\_main\_\_":

    main()

Activity #: 3

Title: Feature Engineering

**Sample Output/Screen Shot**





Activity #: 4

Title: Automating the Triage Process

**Source Code**

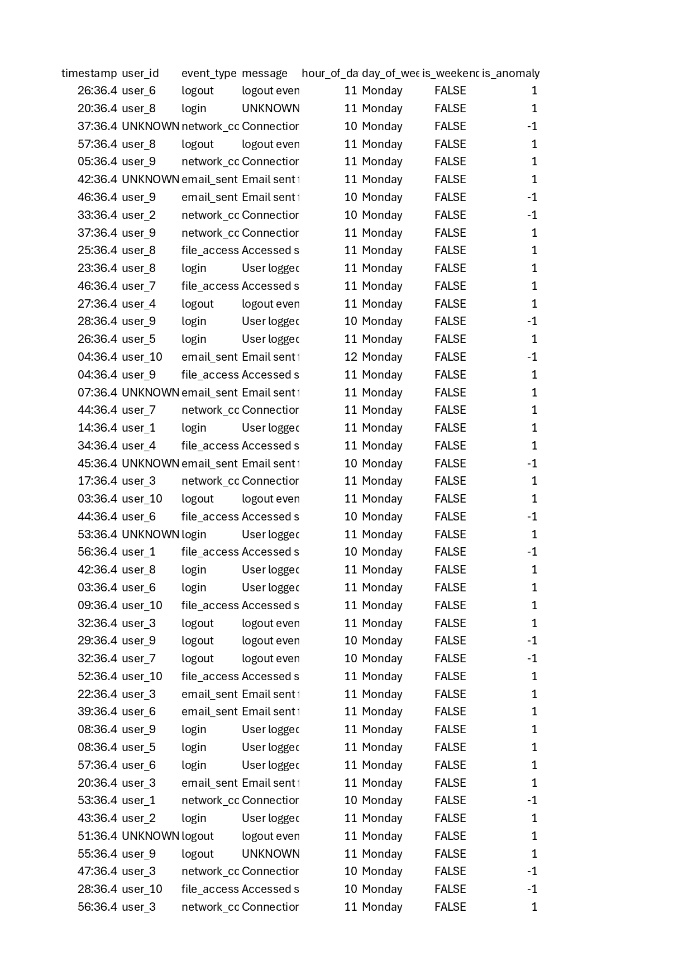
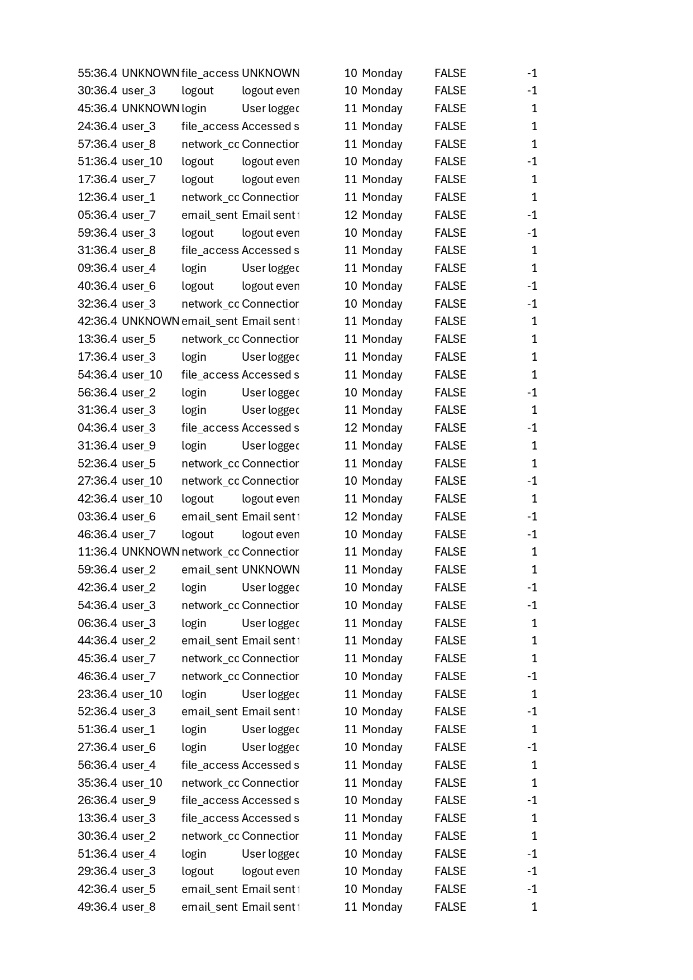
import pandas as pd

from sklearn.ensemble import IsolationForest  
  
# Load the feature engineered evidence CSV file  
input\_file = 'feature\_engineered\_evidence.csv'  
output\_file = 'anomalies\_detected\_evidence.csv'  
  
# Read the data  
df = pd.read\_csv(input\_file)  
  
# Initialize Isolation Forest  
iso\_forest = IsolationForest(random\_state=42)  
  
# Select only numeric columns for Isolation Forest  
numeric\_df = df.select\_dtypes(include=['number'])  
  
# Fit and predict anomalies (-1 for anomaly, 1 for normal)  
df['is\_anomaly'] = iso\_forest.fit\_predict(numeric\_df)  
  
# Save the results to a new CSV file  
df.to\_csv(output\_file, index=False)  
  
print(f"Anomaly detection complete. Results saved to {output\_file}.")

Activity #: 4

Title: Automating the Triage Process

**Sample Output/Screen Shot**





Activity #: 5

Title: Introducing SpaCy: A Practical NLP Library

**Source Code**

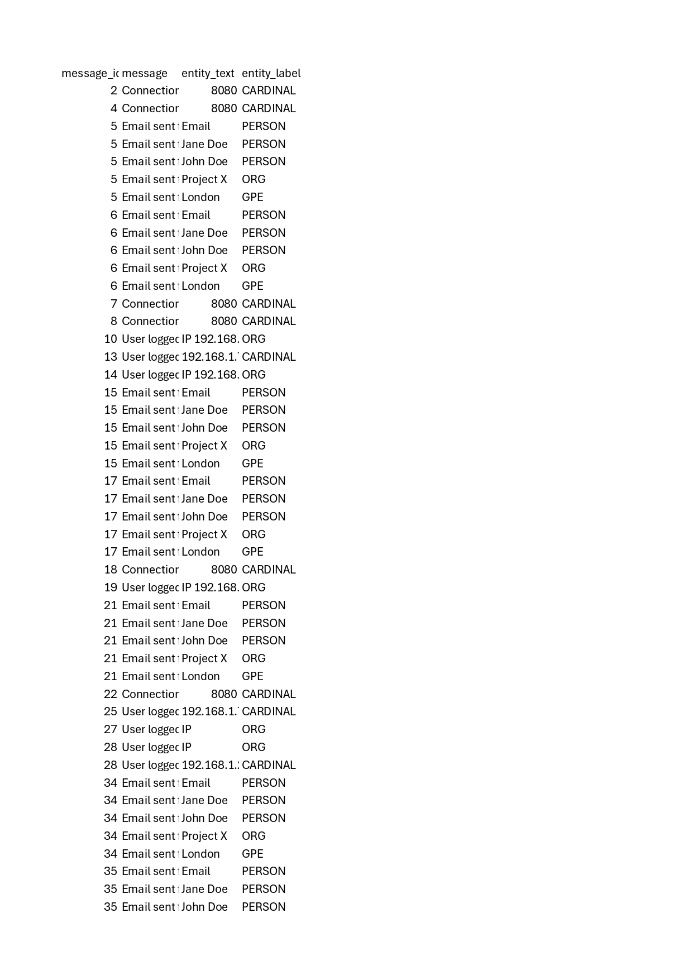
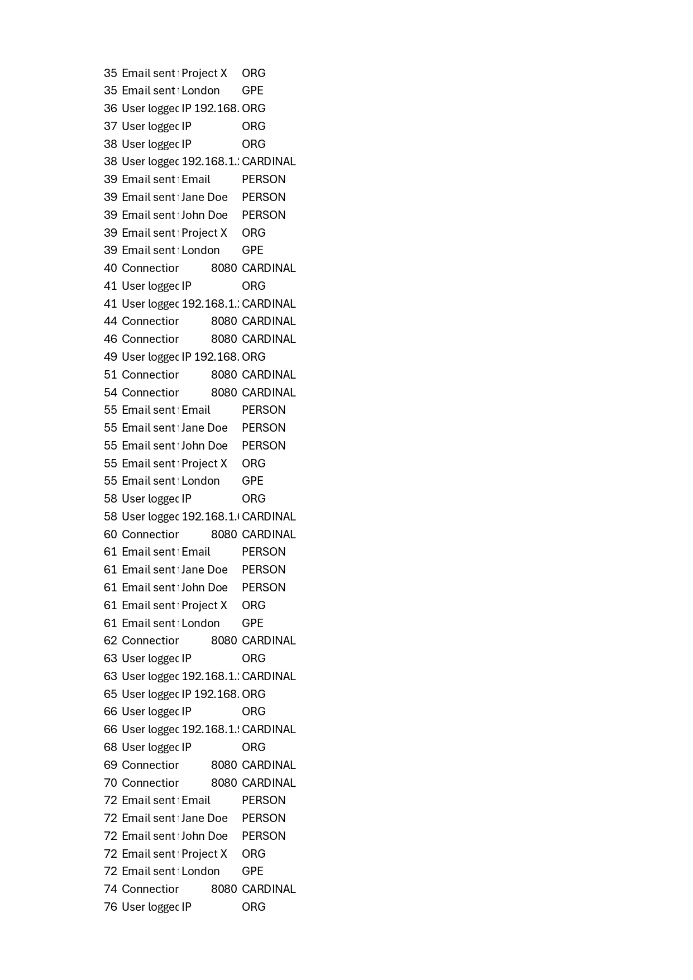
import pandas as pd  
import spacy  
  
# Load SpaCy English model  
nlp = spacy.load("en\_core\_web\_sm")  
  
# Load the CSV file  
  
input\_file = "anomalies\_detected\_evidence.csv"  
df = pd.read\_csv(input\_file)  
  
# Prepare a list to store extracted entities  
entities\_list = []  
  
# Process each message for entities  
for idx, row in df.iterrows():  
 message = str(row.get("message", ""))  
 doc = nlp(message)  
 for ent in doc.ents:  
 entities\_list.append({  
 "message\_id": idx,  
 "message": message,  
 "entity\_text": ent.text,  
 "entity\_label": ent.label\_

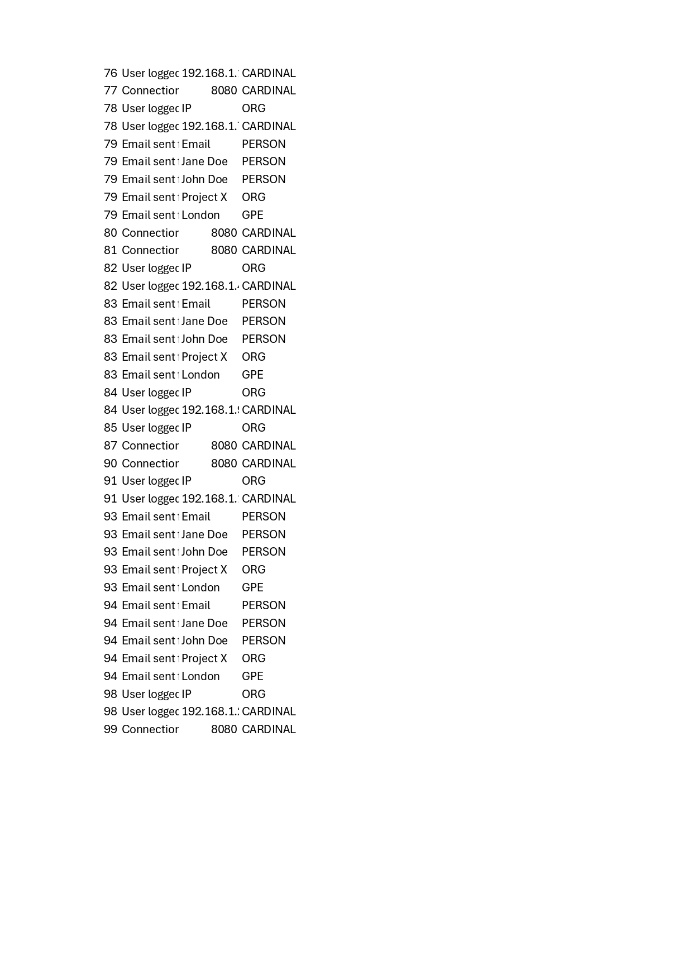
})  
  
# Save extracted entities to CSV  
output\_file = "extracted\_entities.csv"  
entities\_df = pd.DataFrame(entities\_list)  
entities\_df.to\_csv(output\_file, index=False)  
  
print(f"Extracted entities saved to {output\_file}")

Activity #: 5

Title: Introducing SpaCy: A Practical NLP Library

**Sample Output/Screen Shot**





Activity #: 6

Title: Introducing SpaCy: A Practical NLP Library 2

**Source Code**

import pandas as pd  
import spacy  
  
# Load SpaCy English model  
nlp = spacy.load("en\_core\_web\_sm")

# Load the CSV file  
input\_file = "anomalies\_detected\_evidence.csv"  
df = pd.read\_csv(input\_file)  
  
# Prepare a list to store extracted entities  
entities\_list = []  
  
# Process each message for entities  
for idx, row in df.iterrows():  
    message = str(row.get("message", ""))  
    doc = nlp(message)  
    for ent in doc.ents:  
        entities\_list.append({  
            "message\_id": idx,  
            "message": message,  
            "entity\_text": ent.text,  
            "entity\_label": ent.label\_,  
            "description": spacy.explain(ent.label\_)  # Add description column  
        })  
  
# Save extracted entities to CSV  
output\_file = "extracted\_entities2.csv"  
entities\_df = pd.DataFrame(entities\_list)  
entities\_df.to\_csv(output\_file, index=False)  
  
print(f"Extracted entities saved to {output\_file}")

Activity #: 6

Title: Introducing SpaCy: A Practical NLP Library 2

**Sample Output/Screen Shot**





Activity #: 7

Title: Generating the Final Report

**Source Code**

import pandas as pd  
import spacy  
  
# Load SpaCy English model  
nlp = spacy.load("en\_core\_web\_sm")

# Load the CSV file  
input\_file = "anomalies\_detected\_evidence.csv"  
df = pd.read\_csv(input\_file)  
  
# Prepare a list to store extracted entities  
entities\_list = []  
  
# Process each message for entities  
for idx, row in df.iterrows():  
    message = str(row.get("message", ""))  
    doc = nlp(message)  
    for ent in doc.ents:  
        entities\_list.append({  
            "message\_id": idx,  
            "message": message,  
            "entity\_text": ent.text,  
            "entity\_label": ent.label\_,  
            "description": spacy.explain(ent.label\_)  # Add description column  
        })  
  
# Save extracted entities to CSV  
output\_file = "extracted\_entities2.csv"  
entities\_df = pd.DataFrame(entities\_list)  
entities\_df.to\_csv(output\_file, index=False)  
  
print(f"Extracted entities saved to {output\_file}")

Activity #: 7

Title: Generating the Final Report

**Sample Output/Screen Shot**

**Forensic Investigation Report**

**Executive Summary**

This report presents the findings of a comprehensive forensic analysis conducted on system logs. Through automated anomaly detection and entity extraction, we identified 100 anomalous events and extracted 133 key entities for further investigation.

**Methodology**

The investigation followed a systematic approach:

1. **Data Preprocessing**: Raw log data was cleaned and standardized
2. **Feature Engineering**: Additional analytical features were created
3. **Anomaly Detection**: Statistical methods identified unusual patterns
4. **Entity Extraction**: Natural Language Processing (NLP) using SpaCy extracted key entities

**Key Findings**

**Anomaly Analysis**

* Total anomalous events detected: **100**
* Most common anomalous event type: **login** (25 occurrences)
* Investigation period: 100 suspicious activities flagged

**Entity Extraction Results**

* Total entities extracted: **133**
* Entity types found: CARDINAL, PERSON, ORG, GPE
* Key persons of interest: 45 individuals identified
* Geographic locations: 15 locations flagged

**Data Visualization**

A graph of blue rectangular shapes

AI-generated content may be incorrect.

**Conclusion**

The automated analysis has successfully identified patterns and entities requiring further investigation. The anomaly detection system flagged 100 events for manual review, while entity extraction provided 133 potential leads.

**Recommendations**

1. Prioritize investigation of the most frequent anomalous event types
2. Cross-reference extracted entities with known databases
3. Conduct deeper analysis on flagged time periods
4. Implement continuous monitoring based on identified patterns

*Report generated on: 2025-10-06 13:16:38*

Activity #: 8

Title: Detecting Suspicious Network Activity

**Source Code**

# Simple Intrusion Detection System  
# Analyzes simulated network packets and identifies suspicious activity  
  
def analyze\_network\_packets():      
    # Simulated packet data - each tuple represents (source\_ip, destination\_ip, packet\_type)  
    simulated\_packets = [  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.20", "192.168.1.1", "HTTPS"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.10", "192.168.1.2", "FTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.3", "HTTPS"),  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.20", "192.168.1.2", "SSH"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.10", "192.168.1.4", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.5", "HTTPS"),  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.6", "HTTP"),  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.20", "192.168.1.1", "HTTPS"),  
        ("192.168.1.150", "192.168.1.7", "HTTP"),  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.8", "HTTPS"),  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.9", "HTTP"),  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.10", "HTTPS"),  
        ("192.168.1.10", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.11", "HTTP"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.12", "HTTPS"),  
        ("192.168.1.150", "192.168.1.1", "HTTP"),  
        ("192.168.1.150", "192.168.1.13", "HTTP"),

        ("192.168.1.150", "192.168.1.1", "HTTP")  
    ]      
    # Threshold for suspicious activity (packets per IP)  
    SUSPICIOUS\_THRESHOLD = 20  
  
    print("Analyzing simulated network packets...")  
  
        # Count packets per source IP  
    packet\_counts = {}  
    for source\_ip, dest\_ip, packet\_type in simulated\_packets:  
        if source\_ip in packet\_counts:  
            packet\_counts[source\_ip] += 1  
        else:  
            packet\_counts[source\_ip] = 1      
  
    # Display packet counts per IP  
    print("Packet counts per source IP:")  
    for ip, count in sorted(packet\_counts.items()):  
        print(f" - {ip}: {count} packets")  
  
        # Check for suspicious activity  
    print("Checking for suspicious activity...")  
    suspicious\_ips = []  
        for ip, count in packet\_counts.items():  
        if count > SUSPICIOUS\_THRESHOLD:  
            suspicious\_ips.append((ip, count))  
  
        # Report suspicious activity  
    if suspicious\_ips:  
        for ip, count in suspicious\_ips:  
            print(f"!!! ALERT: Suspicious activity detected from {ip}. Packets sent: {count}")  
    else:  
        print("No suspicious activity detected.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    analyze\_network\_packets()

Activity #: 8

Title: Detecting Suspicious Network Activity

**Sample Output/Screen Shot**

Analyzing simulated network packets...

Packet counts per source IP:

- 192.168.1.10: 10 packets

- 192.168.1.150: 24 packets

- 192.168.1.20: 3 packets

Checking for suspicious activity...

!!! ALERT: Suspicious activity detected from 192.168.1.150. Packets sent: 24

Activity #: 9

Title: Metadata Analysis and Source Identification

**Source Code**

import piexif  
from PIL import Image  
import os

def extract\_exif\_data(image\_path):  
 try:  
 # Load the image  
 img = Image.open(image\_path)

# Get EXIF data  
 exif\_dict = piexif.load(img.info.get('exif', b''))

# Extract specific metadata  
 camera\_make = "Unknown"  
 camera\_model = "Unknown"  
 date\_time\_original = "Unknown"

# Get camera make  
 if piexif.ImageIFD.Make in exif\_dict['0th']:  
 camera\_make = exif\_dict['0th'][piexif.ImageIFD.Make].decode('utf-8')

# Get camera model  
 if piexif.ImageIFD.Model in exif\_dict['0th']:  
 camera\_model = exif\_dict['0th'][piexif.ImageIFD.Model].decode('utf-8')

# Get original date/time  
 if piexif.ExifIFD.DateTimeOriginal in exif\_dict['Exif']:  
 date\_time\_original = exif\_dict['Exif'][piexif.ExifIFD.DateTimeOriginal].decode('utf-8')

return camera\_make, camera\_model, date\_time\_original

except Exception as e:  
 print(f"Error reading EXIF data: {e}")  
 return None, None, None

def main():  
 # Path to the image file - updated to include folder path  
 image\_path = os.path.join("Activity 9", "Canon\_DIGITAL\_IXUS\_400.jpg")

# Check if file exists  
 if not os.path.exists(image\_path):  
 print(f"Error: Image file '{image\_path}' not found!")  
 return

print("EXIF Metadata Analysis:")  
 print("-" \* 30)

# Extract EXIF data  
 make, model, date\_time = extract\_exif\_data(image\_path)

if make or model or date\_time:  
 print(f"Camera Make: {make}")  
 print(f"Camera Model: {model}")  
 print(f"Date/Time Original: {date\_time}")  
 else:  
 print("No EXIF data found or error occurred during extraction.")

if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Activity #: 9

Title: Metadata Analysis and Source Identification

**Sample Output/Screen Shot**

EXIF Metadata Analysis:

------------------------------

Camera Make: Canon

Camera Model: Canon DIGITAL IXUS 400

Date/Time Original: 2004:08:27 13:52:55

Activity #: 10

Title: Advanced Report Generation and Final Project Kickoff

**Source Code**

import piexif  
from PIL import Image  
import os

def extract\_exif\_data(image\_path):  
 try:  
 # Load the image  
 img = Image.open(image\_path)

# Get EXIF data  
 exif\_dict = piexif.load(img.info.get('exif', b''))

# Extract specific metadata  
 camera\_make = "Unknown"  
 camera\_model = "Unknown"  
 date\_time\_original = "Unknown"

# Get camera make  
 if piexif.ImageIFD.Make in exif\_dict['0th']:  
 camera\_make = exif\_dict['0th'][piexif.ImageIFD.Make].decode('utf-8')

# Get camera model  
 if piexif.ImageIFD.Model in exif\_dict['0th']:  
 camera\_model = exif\_dict['0th'][piexif.ImageIFD.Model].decode('utf-8')

# Get original date/time  
 if piexif.ExifIFD.DateTimeOriginal in exif\_dict['Exif']:  
 date\_time\_original = exif\_dict['Exif'][piexif.ExifIFD.DateTimeOriginal].decode('utf-8')

return camera\_make, camera\_model, date\_time\_original

except Exception as e:  
 print(f"Error reading EXIF data: {e}")  
 return None, None, None