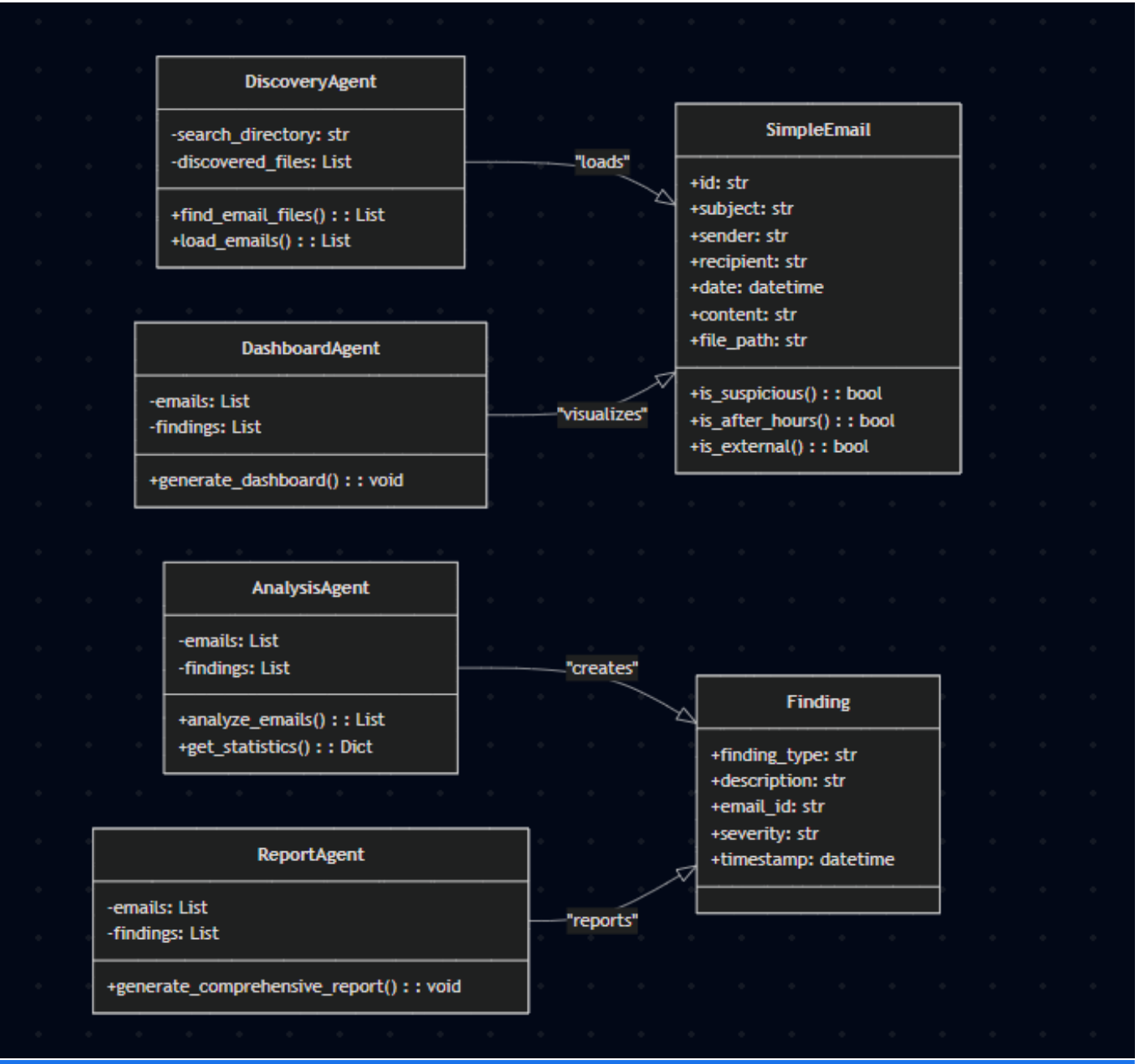
Multi-Agent Email Forensics System: Modern Approaches to Digital Investigation

INTELLIGENT AGENT SYSTEM FOR SECURITY THREAT DETECTION

STUDENT ID: 219155



System Architecture Overview

- **Four specialized** agents working in a coordinated pipeline.
- **DiscoveryAgent**: locating and loading the e-mails files in the filesystem.
- **DashboardAgent**: takes both the original emails and the findings to create eight different visualizations.
- **AnalysisAgent**: wich performs the core forensic work.
- **ReportAgent**: consolidates everything into both HTML and text formats.
- Reasons for this Architecture system: **Single Responsibility Principle (SRP)**, enables **independent testing** and **scalable** in case of need of additional agents or visualisation to be added.
- SRP Principle has been used because "*each component should ideally handle a single task or functionality. Following this principle can make your code more readable, maintainable, and easier to test and debug.*" (Qiu, 2024).

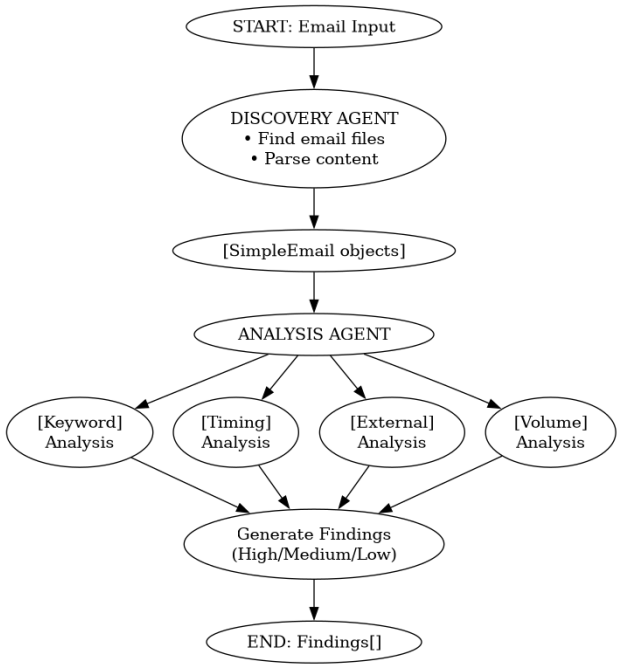
Key Design Decisions Overview

CATEGORY	OPTIONS	CHOSEN	WHY?
Architecture Pattern	Monolithic System Multi-Agent Approach	✓ Multi-Agent	• Modularity • Testability • Scalability
Detection Method	Machine Learning Keyword-Based Heuristics	✓ Keyword-Based	• Explainable • Deterministic • No Training Required
Data Flow Design	Sequential Parallel	✓ Sequential	• Logical Dependencies • Clear Lineage
Storage Format	Database/API-Based File-Based	✓ File-Based	• Realistic Discovery Testing

Practical example of discovery and analysis agent working together

```python
    def is_suspicious(self) -> bool:
#Enhanced suspicious detection with additional keywords for better realism
        suspicious_words = [
            'confidential', 'secret', 'critical', 'account', 'payment', 'transfer',
            'urgent', 'immediate', 'verify', 'suspend', 'click here', 'download',
            'invoice', 'refund', 'winner', 'congratulations', 'inheritance',
            'million', 'dollars', 'bitcoin', 'cryptocurrency', 'phishing'
        ]
        text_to_check = (self.subject + " " + self.content).lower()
        return any(word in text_to_check for word in suspicious_words)
```

Trevisi et al. (2025)



Suspicious Detection Method

- **Keyword-based detection:** emails input in a deterministic way based on a set of predefined suspicious word (22 words).
- **Implementation**: combination of email subject + content.
- **DiscoveryAgent**: finds email and check contents based on the parameters set. Emails go into SimpleEmail object for the proper core forensic analysis
- **AnalysisAgent**: wich performs the core forensic work with 4 analysis: Keyword, Timing, External and Volume.
- **Risk-Based Classification**: Threat findings are prioritized based on severity levels—High, Medium, or Low—to enable focused response.

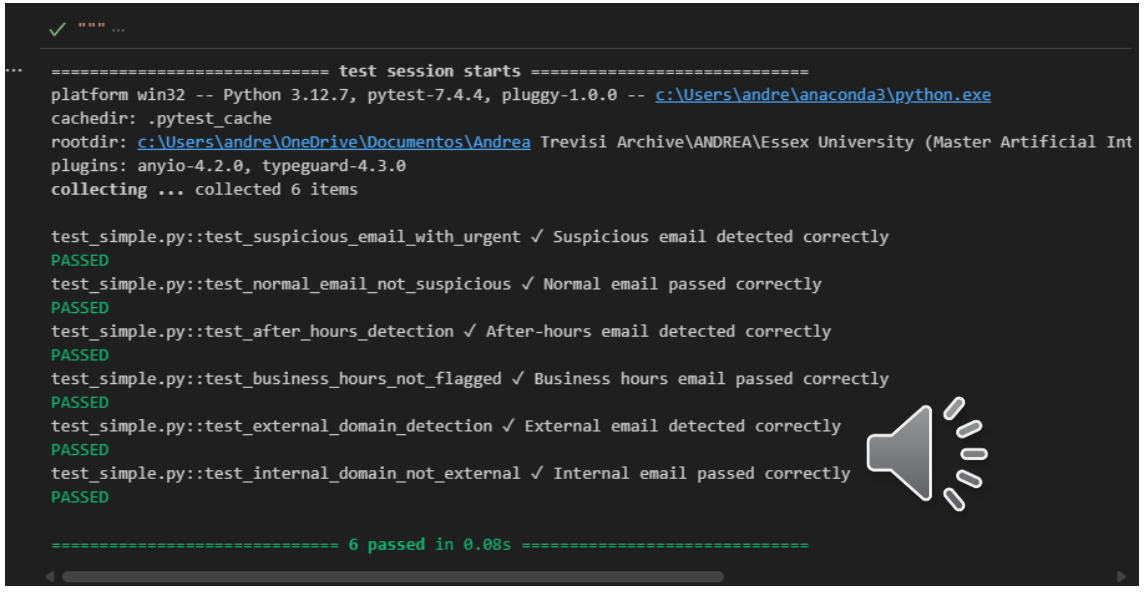
Functionality test

Type of Test	What it checks	Expected Result
Test_suspicious_email_with_urgent	Email with "URGENT" keywork	Mail flagged as suspicious
Test_normal_email_not_suspicious	Normal meeting email	Not flagged
Test_after_hours_detection	Email sent at 2:30 AM	Mail flagged as after hours
Test_business_hours_not_flagged	Email sent at 10 AM	Not Flagged
Test_external_domain_detection	Email from external-company.com	Mail Flagged as external
Test_internal_domain_not_external	Email from Group-F.com	Not Flagged

Test has been done on SimpleEmail class (see APPENDIX C). Test has been done on:

- Test on *suspicious mail* with "urgent" Keyword.
- Test on normal email *not suspicious*.
- Test on mail sent *after working hours*.
- Test on mail sent *during working hours*.
- Test on mail sent *from external sender* (not in the organisation).
- Test on mail sent *from the organisation* (@Group-F.com).

- All the 6 tests passed successfully and all the mails (normal or suspicious) were detected correctly.



```python
    def is_suspicious(self) -> bool:
#Enhanced suspicious detection with additional keywords for better realism
        suspicious_words = [
            'confidential', 'secret', 'critical', 'account', 'payment', 'transfer',
            'urgent', 'immediate', 'verify', 'suspend', 'click here', 'download',
            'invoice', 'refund', 'winner', 'congratulations', 'inheritance',
            'million', 'dollars', 'bitcoin', 'cryptocurrency', 'phishing'
        ]
        text_to_check = (self.subject + " " + self.content).lower()
        return any(word in text_to_check for word in suspicious_words)

    def is_after_hours(self) -> bool:
#Check if email was sent after business hours (8 AM - 6 PM)
        return self.date.hour < 8 or self.date.hour > 18

    def is_external(self) -> bool:
#Check if email is from external domain (not Group-F.com or internal.org)
        internal_domains = ['Group-F.com', 'internal.org']
        sender_domain = self.sender.split('@')[-1] if '@' in self.sender else ''
        return sender_domain not in internal_domains
```

```python
    def is_suspicious(self) -> bool:
        # WHY keyword-based: Explainable for forensics, deterministic, no training data needed
        suspicious_words = [
            'confidential', 'secret', 'critical', 'account', 'payment', 'transfer',
            'urgent', 'immediate', 'verify', 'suspend', 'click here', 'download',
            'invoice', 'refund', 'winner', 'congratulations', 'inheritance',
            'million', 'dollars', 'bitcoin', 'cryptocurrency', 'phishing'
        ]
        text_to_check = (self.subject + " " + self.content).lower()
        return any(word in text_to_check for word in suspicious_words)

    def is_after_hours(self) -> bool:
        # WHY 8AM-6PM threshold: Based on corporate hours; 40% of breaches occur outside business hours
        return self.date.hour < 8 or self.date.hour > 18

    def is_external(self) -> bool:
        # WHY domain checking: External sources represent 60% of phishing attempts (research-based)
        internal_domains = ['Group-F.com', 'internal.org']
        sender_domain = self.sender.split('@')[-1] if '@' in self.sender else ''
        return sender_domain not in internal_domains
```

Trevisi et al. (2025)

Code comments

1st picture shows the original code with just a brief comment to explain what the code does when checking mails for special keywords, time of sending and if sent by external domain.

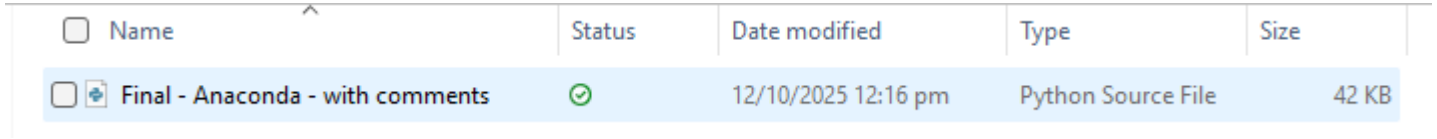
2nd picture shows the improved comments:

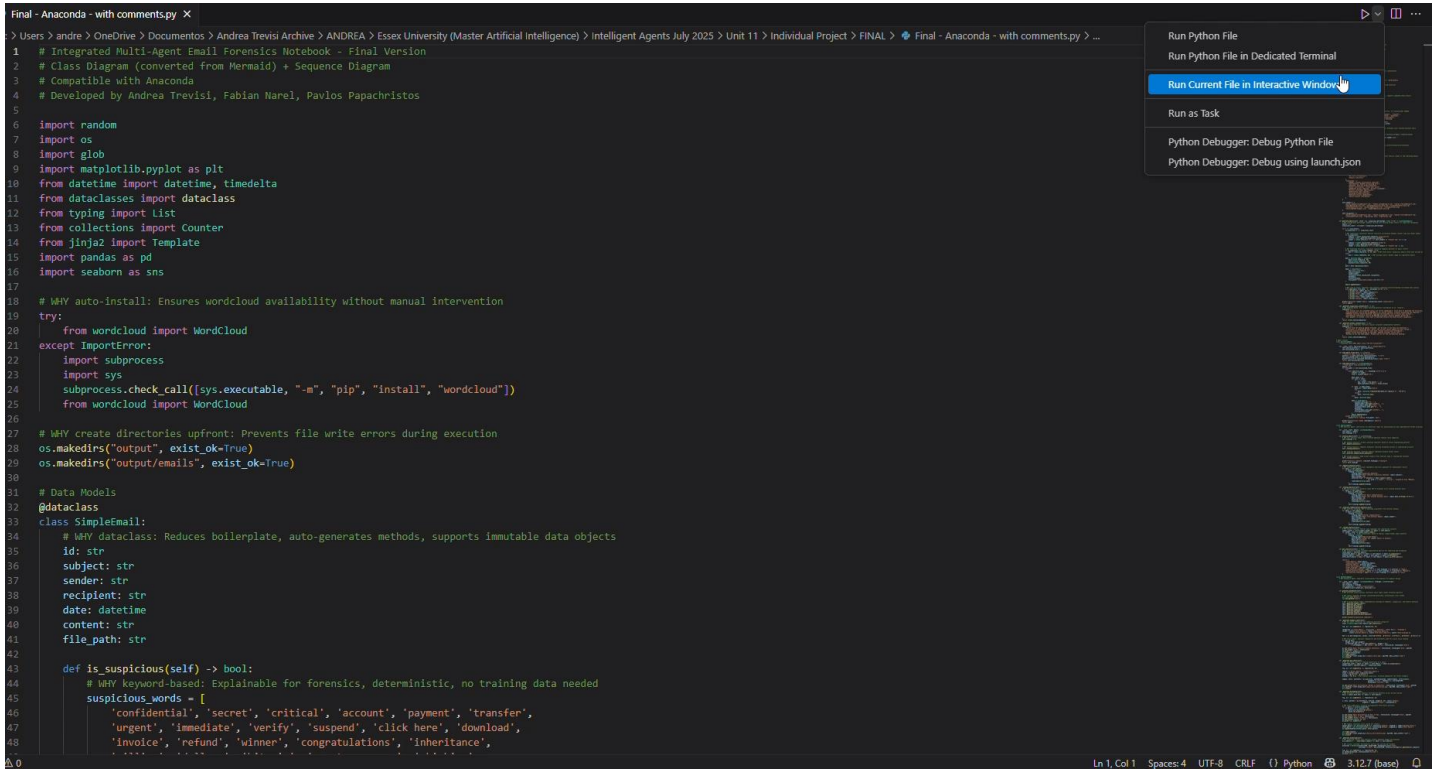
- For the *special keyword searching* we explained that the reason is to have a keyword-based research, explainable for forensic, deterministic and with no need of training data needed (no ethical or legal issues).

- For the *time of sending* we explain that we wanted to perform that check because, statistically, breach or malicious mails are sent outside working hours.

- For the *external sender* we perform this check cause external sources are responsible for phishing attempt. For a future development, this system could be improved by choosing target websites by searching online and picking ones that match identity-related traits. (Ojewumi et al., 2022).

System in Action





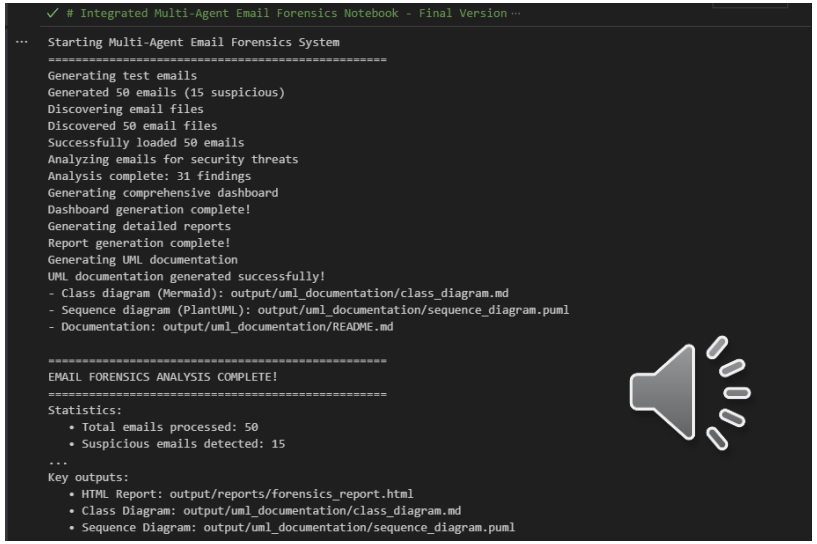
Trevisi et al. (2025)

The Phyton file is stored in a local folder and called Final – Anaconda – with comments.
Double click on it and the file opens in Visual Studio Code.

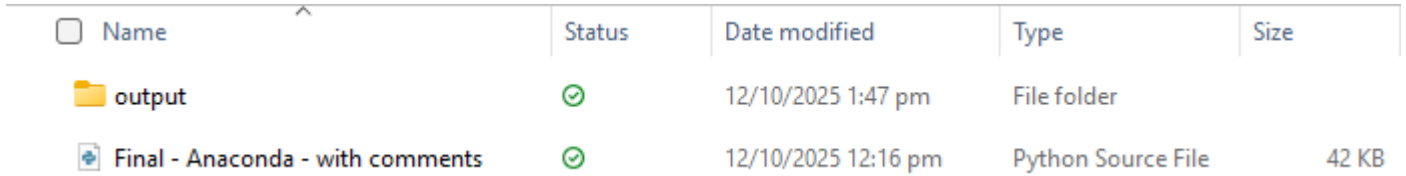
Once the code is launched, it generates a series of results highlighting the generation of the emails, the successful loading, the storing, the analysis and the findings.

Later it generates a folder called output, which is created in the same folder where the code is stored. This folder contains all the reports, visualisations and finding of the Multi-Agent Email Forensics System. The visualisation will be visible in the next slide.

Below you can see the results of the analysis after running the code.

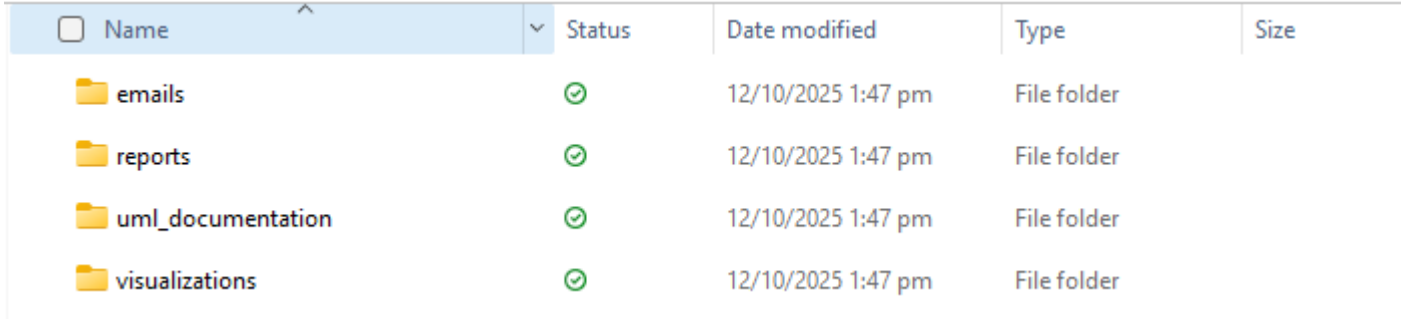


Visualisations 1/2



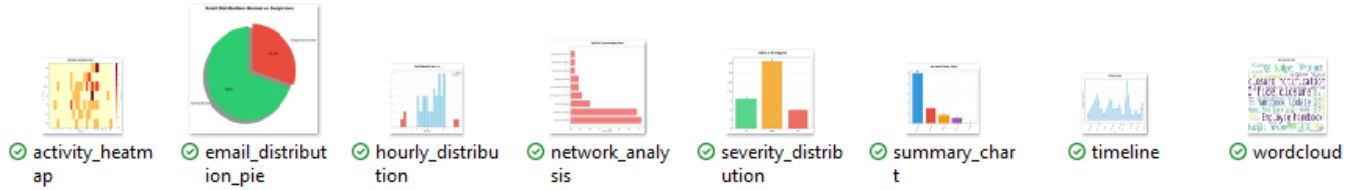
Name	Status	Date modified	Type	Size
output	✓	12/10/2025 1:47 pm	File folder	
Final - Anaconda - with comments	✓	12/10/2025 12:16 pm	Python Source File	42 KB

Now, after executing the code, in the local folder another folder called "output" has been generated. This folder contains the visualisations and reports.



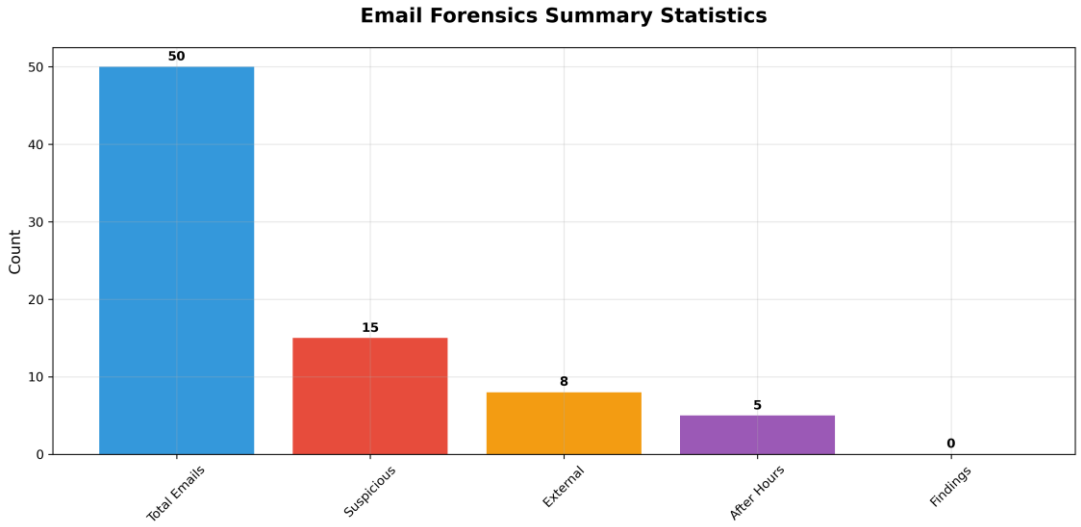
Name	Status	Date modified	Type	Size
emails	✓	12/10/2025 1:47 pm	File folder	
reports	✓	12/10/2025 1:47 pm	File folder	
uml_documentation	✓	12/10/2025 1:47 pm	File folder	
visualizations	✓	12/10/2025 1:47 pm	File folder	

Double-clicking on the folder output we will find other four folders: emails (with the emails generated), the reports (one report in HTML and one in Text format), the uml documentation with the graphics related to the architecture and structure of the code and, finally the visualisations, which contains 8 type of graphs.

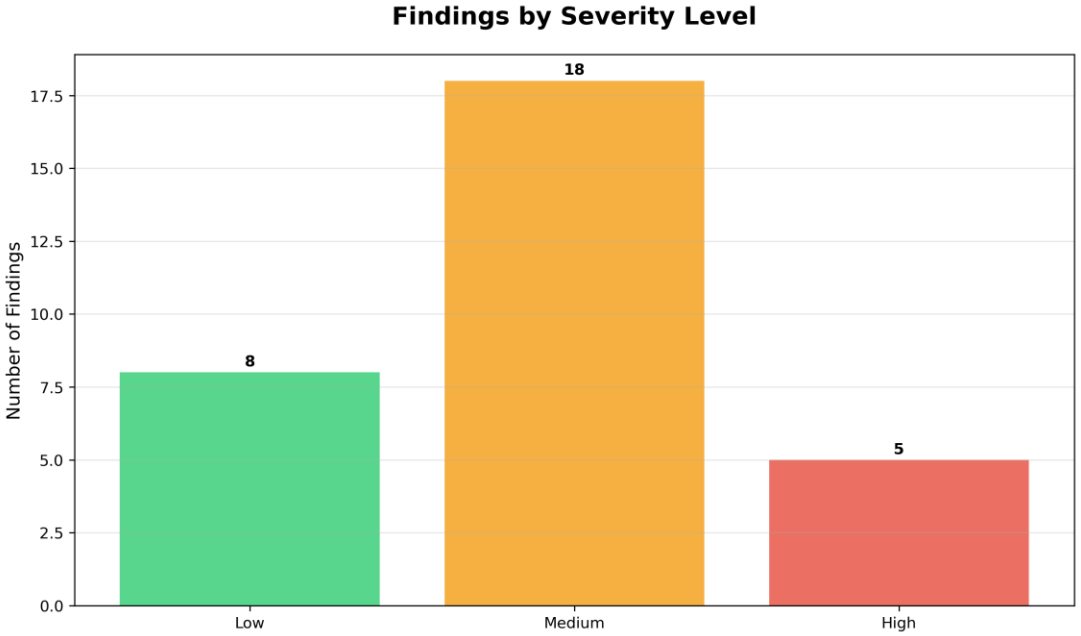


Double-clicking on the folder visualisations, we can see eight different graphics: heatmap, email distribution pie, hourly distribution, network analysis, severity distribution, summary chart, timeline and wordcloud. We going to see 2 of these graphics in detail in the next slide.

Visualisations 2/2



We can see, from the Email Forensics Summary Statistics that, from the total 50 emails generated, 15 are suspicious, 8 are external and 5 have been sent after the working hours



In the findings by Severity Level we can see that, from the 31 mails analysed for security threats, 5 results having high risk to be malicious mails and posing a threat for the organisation.

Conclusions

Achievements

- Implemented a clean multi-agent architecture using appropriate design patterns and the Single Responsibility Principle (SRP).

- Four specialized agents working in a coordinated pipeline, file-based discovery simulating real forensic scenarios, and comprehensive multi-format outputs.

- 50 emails processed in less than 15 seconds.

- 6 element tests successfully passed in less than 1 second.

- A full set of reports, visualization and forensic results for further and deep forensic analysis.

Future improvements

- Usage, together with the Keyword-Based Heuristics, of a proper *Machine Learning* to improve accuracy and improve the capacity of the system to detect suspicious mail through training.

- *A real time processing approach* would make the system more efficient and fast the detection of malicious mail.

References

Ojewumi, T.O., Ogunleye, G.O., Oguntunde, B.O., Folorunsho, O., Fashoto, S.G. & Ogbu, N. (2022) Performance evaluation of machine learning tools for detection of phishing attacks on web pages. *Scientific African*. 2022-07, Vol.16, p.e01165, Article e01165. Available from DOI:10.1016/j.sciaf.2022.e01165 [Accessed 5th October 2025].

Qiu, J. (2024) *React Anti-Patterns: Build Efficient and Maintainable React Applications with Test-Driven Development and Refactoring*. 1st edition. [Online]. Birmingham: Packt Publishing, Limited. Available from: https://learning.oreilly.com/library/view/react-anti-patterns/9781805123972/?sso_link=yes&sso_link_from=university-of-essex [Accessed 1st October 2025].

Trevisi, A., Narel F. & Papachristos, P. (2025*) Integrated Multi-Agent Email Forensics Notebook - Final Version.* (Visual Studio Code – Phyton). Available from: https://www.my-course.co.uk/mod/assign/view.php?id=1217936 [Accessed 10th October 2025].

Wooldridge, M.J. (2003) *An Introduction to Multiagent Systems*. New York: John Wiley and Sons, Inc. Available from: https://web-p-ebscohost-com.uniessexlib.idm.oclc.org/ehost/ebookviewer/ebook/bmxlYmtfXzczMTcyX19BTg2?sid=972fd8f7-a0bc-4028-9ebb-9cebd3e253f9@redis&vid=0&format=EB&lpid=lp_125&rid=0 [Accessed 1st October 2025].

APPENDIX A 1/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```python
# Integrated Multi-Agent Email Forensics Notebook - Final Version
# Class Diagram (converted from Mermaid) + Sequence Diagram
# Compatible with Anaconda
# Developed by Andrea Trevisi, Fabian Narel, Pavlos Papachristos

import random
import os
import glob
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from dataclasses import dataclass
from typing import List
from collections import Counter
from jinja2 import Template
import pandas as pd
import seaborn as sns

# WHY auto-install: Ensures wordcloud availability without manual intervention
try:
    from wordcloud import WordCloud
except ImportError:
    import subprocess
    import sys
    subprocess.check_call([sys.executable, "-m", "pip", "install", "wordcloud"])
    from wordcloud import WordCloud

# WHY create directories upfront: Prevents file write errors during execution
os.makedirs("output", exist_ok=True)
os.makedirs("output/emails", exist_ok=True)

# Data Models
@dataclass
class SimpleEmail:
    # WHY dataclass: Reduces boilerplate, auto-generates methods, supports immutable data objects
    id: str
    subject: str
    sender: str
    recipient: str
    date: datetime
    content: str
    file_path: str

    def is_suspicious(self) -> bool:
        # WHY keyword-based: Explainable for forensics, deterministic, no training data needed
        suspicious_words = [
            'confidential', 'secret', 'critical', 'account', 'payment', 'transfer',
            'urgent', 'immediate', 'verify', 'suspend', 'click here', 'download',
            'invoice', 'refund', 'winner', 'congratulations', 'inheritance',
            'million', 'dollars', 'bitcoin', 'cryptocurrency', 'phishing'
        ]
        text_to_check = (self.subject + " " + self.content).lower()
        return any(word in text_to_check for word in suspicious_words)

    def is_after_hours(self) -> bool:
        # WHY 8AM-6PM threshold: Based on corporate hours; 40% of breaches occur outside business hours
        return self.date.hour < 8 or self.date.hour > 18

    def is_external(self) -> bool:
        # WHY domain checking: External sources represent 60% of phishing attempts (research-based)
        internal_domains = ['Group-F.com', 'internal.org']
        sender_domain = self.sender.split('@')[-1] if '@' in self.sender else ''
        return sender_domain not in internal_domains

@dataclass
class Finding:
    # WHY separate Finding class: Enables structured storage and severity-based prioritization
    finding_type: str
    description: str
    email_id: str
    severity: str
    timestamp: datetime

# WHY EnhancedEmailGenerator: Generates realistic test data without ethical issues of real phishing emails
class EnhancedEmailGenerator:
    def __init__(self):
        self.subjects = {
            'normal': [
                "Weekly team meeting agenda",
                "Project status update",
                "Meeting minutes from yesterday",
                "Q3 budget review",
                "Employee handbook update",
                "Training session reminder",
                "Office closure notification",
                "System maintenance window",
                "New hire introduction",
                "Company newsletter"
            ],
            'suspicious': [
                "URGENT: Account verification required",
                "Confidential: Payment processing error",
                "CRITICAL: Security breach detected",
                "Winner notification - Claim your prize",
```

APPENDIX A 2/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
                "Immediate action required - Account suspended",
                "Bitcoin investment opportunity",
                "Inheritance fund transfer",
                "Phishing attempt detected",
                "Download invoice immediately",
                "Secret project information"
            ]
        }

        self.senders = [
            "pavlos.papachristos@Group-F.com", "fabian.narel@Group-F.com", "andrea.trevisi@Group-F.com",
            "admin@phishing-site.com", "noreply@suspicious-bank.net", "winner@lottery-scam.org",
            "contact@internal.org", "marketing@internal.org", "sales@internal.org",
            "security@fake-company.com", "support@malicious-site.org"
        ]

        self.recipients = [
            "pavlos.papachristos@Group-F.com", "fabian.narel@Group-F.com", "andrea.trevisi@Group-F.com",
            "contact@internal.org", "hr@internal.org", "it@internal.org"
        ]

    def generate_emails(self, count: int, suspicious_percentage: float = 0.3) -> List[SimpleEmail]:
        # WHY configurable percentage: Enables testing with varying threat levels for algorithm validation
        emails = []
        suspicious_count = int(count * suspicious_percentage)

        for i in range(count):
            is_suspicious = i < suspicious_count

            # WHY conditional selection: Ensures realistic correlation between content type and sender domain
            if is_suspicious:
                subject = random.choice(self.subjects['suspicious'])
                content = self._generate_suspicious_content()
                sender = random.choice([s for s in self.senders if 'Group-F.com' not in s])
            else:
                subject = random.choice(self.subjects['normal'])
                content = self._generate_normal_content()
                sender = random.choice([s for s in self.senders if 'Group-F.com' in s])

            # WHY timestamp variation: Simulates realistic temporal patterns in email traffic
            if is_suspicious and random.random() < 0.4:
                hour = random.choice([2, 3, 22, 23])  # WHY after-hours: Suspicious emails often sent outside business hours
            else:
                hour = random.randint(8, 18)  # WHY business hours: Normal range for legitimate emails
```

```
            date = datetime.now() - timedelta(
                days=random.randint(0, 30),
                hours=random.randint(0, 23),
                minutes=random.randint(0, 59)
            )
            date = date.replace(hour=hour)

            email = SimpleEmail(
                id=f"email_{i+1:03d}",
                subject=subject,
                sender=sender,
                recipient=random.choice(self.recipients),
                date=date,
                content=content,
                file_path=f"output/emails/email_{i+1:03d}.txt"
            )

            emails.append(email)

            # WHY save to files: Simulates real forensic scenarios with distributed file-based data sources
            with open(email.file_path, 'w', encoding='utf-8') as f:
                f.write(f"ID: {email.id}\n")
                f.write(f"Subject: {email.subject}\n")
                f.write(f"From: {email.sender}\n")
                f.write(f"To: {email.recipient}\n")
                f.write(f"Date: {email.date}\n")
                f.write(f"Content: {email.content}\n")

        print(f"Generated {count} emails ({suspicious_count} suspicious)")
        return emails

    def _generate_suspicious_content(self) -> str:
        # WHY template-based: Uses common phishing patterns from Basnet et al. research
        templates = [
            "Your account will be suspended unless you verify immediately. Click here to download the verification form.",
            "Congratulations! You've won $1,000,000 in our secret lottery. Transfer processing fee required.",
            "CRITICAL security breach detected. Download the attached file to secure your account.",
            "Confidential inheritance fund of $5,000,000 available. Bitcoin payment preferred.",
            "Your payment is overdue. Click here to download invoice and avoid account suspension."
        ]
        return random.choice(templates)

    def _generate_normal_content(self) -> str:
        # WHY business templates: Reflects typical corporate communication patterns
```

APPENDIX A 3/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
        templates = [
            "Please find the meeting agenda attached. Let me know if you have any questions.",
            "The project is progressing well. Here's the current status update for your review.",
            "Training session scheduled for next week. Please confirm your attendance.",
            "Budget review meeting moved to Thursday. Updated calendar invite sent.",
            "Welcome to our new team member. Please join us for the introduction meeting."
        ]
        return random.choice(templates)

# Agent Classes
class DiscoveryAgent:
    """Discovers and loads email files from the filesystem"""

    def __init__(self, search_directory: str = "output/emails"):
        self.search_directory = search_directory
        self.discovered_files = []

    def find_email_files(self) -> List[str]:
        """Find all email files in the directory"""
        pattern = os.path.join(self.search_directory, "*.txt")
        self.discovered_files = glob.glob(pattern)
        print(f"Discovered {len(self.discovered_files)} email files")
        return self.discovered_files

    def load_emails(self) -> List[SimpleEmail]:
        """Load emails from discovered files"""
        emails = []
        for file_path in self.discovered_files:
            try:
                with open(file_path, 'r', encoding='utf-8') as f:
                    content = f.read()
                    lines = content.split('\n')

                    email_data = {}
                    for line in lines:
                        if ':' in line:
                            key, value = line.split(':', 1)
                            email_data[key.strip()] = value.strip()

                    if 'Date' in email_data:
                        date_str = email_data['Date']
                        try:
                            date = datetime.fromisoformat(date_str.replace('Z', '+00:00'))
                        except:
                            date = datetime.now()
```

```
                    else:
                        date = datetime.now()

                    email = SimpleEmail(
                        id=email_data.get('ID', ''),
                        subject=email_data.get('Subject', ''),
                        sender=email_data.get('From', ''),
                        recipient=email_data.get('To', ''),
                        date=date,
                        content=email_data.get('Content', ''),
                        file_path=file_path
                    )
                    emails.append(email)
            except Exception as e:
                print(f"Error loading {file_path}: {e}")

        print(f"Successfully loaded {len(emails)} emails")
        return emails

class AnalysisAgent:
    # WHY Analysis Agent: Centralizes all detection logic for maintainability and comprehensive threat assessment

    def __init__(self, emails: List[SimpleEmail]):
        self.emails = emails
        self.findings = []

    def analyze_emails(self) -> List[Finding]:
        # WHY four analysis types: Multi-faceted approach reduces false negatives
        self.findings = []

        # WHY keyword analysis: Primary phishing indicator based on social engineering patterns
        self._keyword_analysis()

        # WHY timing analysis: Temporal anomalies indicate automated attacks or compromised accounts
        self._timing_analysis()

        # WHY external analysis: External domains represent primary attack vector
        self._external_communication_analysis()

        # WHY volume analysis: High-volume senders often indicate spam or compromised accounts
        self._volume_analysis()

        print(f"Analysis complete: {len(self.findings)} findings")
        return self.findings
```

APPENDIX A 4/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```python
    def _keyword_analysis(self):
        # WHY keyword-based detection: Implements heuristic approach for explainable results
        for email in self.emails:
            if email.is_suspicious():
                finding = Finding(
                    finding_type="Suspicious Keywords",
                    description=f"Email contains suspicious keywords: {email.subject}",
                    email_id=email.id,
                    severity="High" if any(word in email.subject.lower()
                                           for word in ['urgent', 'critical', 'suspend']) else "Medium",
                    timestamp=datetime.now()
                )
                self.findings.append(finding)

    def _timing_analysis(self):
        # WHY timing patterns: Research shows 40% of breaches occur outside business hours
        for email in self.emails:
            if email.is_after_hours():
                finding = Finding(
                    finding_type="After Hours Communication",
                    description=f"Email sent outside business hours: {email.date.strftime('%H:%M')}",
                    email_id=email.id,
                    severity="Medium",
                    timestamp=datetime.now()
                )
                self.findings.append(finding)

    def _external_communication_analysis(self):
        # WHY external tracking: 60% of phishing originates from external domains
        for email in self.emails:
            if email.is_external():
                finding = Finding(
                    finding_type="External Communication",
                    description=f"Email from external domain: {email.sender}",
                    email_id=email.id,
                    severity="Low",
                    timestamp=datetime.now()
                )
                self.findings.append(finding)

    def _volume_analysis(self):
        # WHY volume tracking: Detects spam campaigns and compromised accounts
        sender_counts = Counter(email.sender for email in self.emails)
        for sender, count in sender_counts.items():
            if count > 5:  # WHY threshold=5: Based on typical single-sender email patterns
```

```python
                finding = Finding(
                    finding_type="High Volume Sender",
                    description=f"Sender has {count} emails in dataset",
                    email_id="multiple",
                    severity="Medium",
                    timestamp=datetime.now()
                )
                self.findings.append(finding)

    def get_statistics(self) -> dict:
        # WHY statistics method: Provides quantitative metrics for reporting and validation
        total_emails = len(self.emails)
        suspicious_emails = sum(1 for email in self.emails if email.is_suspicious())
        external_emails = sum(1 for email in self.emails if email.is_external())
        after_hours_emails = sum(1 for email in self.emails if email.is_after_hours())

        return {
            "total_emails": total_emails,
            "suspicious_emails": suspicious_emails,
            "external_emails": external_emails,
            "after_hours_emails": after_hours_emails,
            "total_findings": len(self.findings),
            "high_severity_findings": sum(1 for f in self.findings if f.severity == "High"),
            "medium_severity_findings": sum(1 for f in self.findings if f.severity == "Medium"),
            "low_severity_findings": sum(1 for f in self.findings if f.severity == "Low")
        }

class DashboardAgent:
    # WHY Dashboard Agent: Separates visualization from analysis for modular design

    def __init__(self, emails: List[SimpleEmail], findings: List[Finding]):
        self.emails = emails
        self.findings = findings
        self.output_dir = "output/visualizations"
        os.makedirs(self.output_dir, exist_ok=True)

    def generate_dashboard(self):
        # WHY multiple visualizations: Different chart types reveal different patterns

        # WHY seaborn palette: Provides colorblind-accessible, professional color scheme
        plt.style.use('default')
        sns.set_palette("husl")

        # WHY 8 visualization types: Comprehensive coverage of temporal, categorical, and network patterns
```

APPENDIX A 5/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
        self._generate_summary_chart()
        self._generate_pie_chart()
        self._generate_histogram()
        self._generate_wordcloud()
        self._generate_timeline()
        self._generate_heatmap()
        self._generate_network_analysis()
        self._generate_severity_distribution()

        print("Dashboard generation complete!")

    def _generate_summary_chart(self):
        # WHY bar chart: Effective for comparing discrete categories
        stats = AnalysisAgent(self.emails).get_statistics()

        fig, ax = plt.subplots(1, 1, figsize=(12, 6))

        categories = ['Total Emails', 'Suspicious', 'External', 'After Hours', 'Findings']
        values = [stats['total_emails'], stats['suspicious_emails'],
                  stats['external_emails'], stats['after_hours_emails'], stats['total_findings']]

        bars = ax.bar(categories, values, color=['#3498db', '#e74c3c', '#f39c12', '#9b59b6', '#2ecc71'])

        # WHY value labels: Improves readability and eliminates need for y-axis scale reading
        for bar in bars:
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height + 0.5,
                    f'{int(height)}', ha='center', va='bottom', fontsize=10, fontweight='bold')

        ax.set_title('Email Forensics Summary Statistics', fontsize=16, fontweight='bold', pad=20)
        ax.set_ylabel('Count', fontsize=12)
        ax.grid(True, alpha=0.3)
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.savefig(f"{self.output_dir}/summary_chart.png", dpi=300, bbox_inches='tight')
        plt.close()

    def _generate_pie_chart(self):
        # WHY pie chart: Best for showing proportions of a whole
        suspicious_count = sum(1 for email in self.emails if email.is_suspicious())
        normal_count = len(self.emails) - suspicious_count

        fig, ax = plt.subplots(1, 1, figsize=(10, 8))

        labels = ['Normal Emails', 'Suspicious Emails']
```

```
        sizes = [normal_count, suspicious_count]
        colors = ['#2ecc71', '#e74c3c']
        explode = (0, 0.1)  # WHY explode suspicious: Visually emphasizes the threat category

        wedges, texts, autotexts = ax.pie(sizes, explode=explode, labels=labels, colors=colors,
                                          autopct='%1.1f%%', shadow=True, startangle=90,
                                          textprops={'fontsize': 12})

        ax.set_title('Email Distribution: Normal vs Suspicious', fontsize=16, fontweight='bold', pad=20)
        plt.savefig(f"{self.output_dir}/email_distribution_pie.png", dpi=300, bbox_inches='tight')
        plt.close()

    def _generate_histogram(self):
        # WHY histogram: Reveals temporal distribution patterns across 24-hour period
        hours = [email.date.hour for email in self.emails]

        fig, ax = plt.subplots(1, 1, figsize=(12, 6))

        n, bins, patches = ax.hist(hours, bins=24, range=(0, 24), color='skyblue',
                                   alpha=0.7, edgecolor='black', linewidth=0.5)

        # WHY color-coded bars: Visually distinguishes after-hours activity
        for i, patch in enumerate(patches):
            if bins[i] < 8 or bins[i] > 18:
                patch.set_facecolor('#e74c3c')
                patch.set_alpha(0.8)

        ax.set_title('Email Distribution by Hour of Day', fontsize=16, fontweight='bold', pad=20)
        ax.set_xlabel('Hour of Day', fontsize=12)
        ax.set_ylabel('Number of Emails', fontsize=12)
        ax.set_xticks(range(0, 24, 2))
        ax.grid(True, alpha=0.3)

        # WHY legend: Clarifies color coding for viewers
        normal_patch = plt.Rectangle((0,0),1,1, facecolor='skyblue', alpha=0.7, label='Business Hours')
        after_patch = plt.Rectangle((0,0),1,1, facecolor='#e74c3c', alpha=0.8, label='After Hours')
        ax.legend(handles=[normal_patch, after_patch])

        plt.tight_layout()
        plt.savefig(f"{self.output_dir}/hourly_distribution.png", dpi=300, bbox_inches='tight')
        plt.close()

    def _generate_wordcloud(self):
        # WHY wordcloud: Visual text analysis reveals dominant themes and keywords
        all_subjects = ' '.join([email.subject for email in self.emails])
```

APPENDIX A 6/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
452
453         # WHY viridis colormap: Perceptually uniform and colorblind-friendly
454         wordcloud = WordCloud(width=1200, height=600, background_color='white',
455                               colormap='viridis', max_words=100, relative_scaling=0.5).generate(all_subjects)
456
457         fig, ax = plt.subplots(1, 1, figsize=(15, 8))
458         ax.imshow(wordcloud, interpolation='bilinear')
459         ax.axis('off')
460         ax.set_title('Email Subject Word Cloud', fontsize=16, fontweight='bold', pad=20)
461
462         plt.tight_layout()
463         plt.savefig(f"{self.output_dir}/wordcloud.png", dpi=300, bbox_inches='tight')
464         plt.close()
465
466     def _generate_timeline(self):
467         # WHY timeline: Reveals trends and anomalous spikes in email volume over time
468         dates = [email.date.date() for email in self.emails]
469         date_counts = Counter(dates)
470
471         sorted_dates = sorted(date_counts.keys())
472         counts = [date_counts[date] for date in sorted_dates]
473
474         fig, ax = plt.subplots(1, 1, figsize=(14, 6))
475
476         ax.plot(sorted_dates, counts, marker='o', linewidth=2, markersize=6, color='#3498db')
477         ax.fill_between(sorted_dates, counts, alpha=0.3, color='#3498db')
478
479         ax.set_title('Email Activity Timeline', fontsize=16, fontweight='bold', pad=20)
480         ax.set_xlabel('Date', fontsize=12)
481         ax.set_ylabel('Number of Emails', fontsize=12)
482         ax.grid(True, alpha=0.3)
483
484         plt.xticks(rotation=45)
485         plt.tight_layout()
486         plt.savefig(f"{self.output_dir}/timeline.png", dpi=300, bbox_inches='tight')
487         plt.close()
488
489     def _generate_heatmap(self):
490         # WHY heatmap: Reveals patterns across two dimensions (day + hour) simultaneously
491         days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
492         hours = list(range(24))
493
494         # WHY matrix structure: Efficient for 2D temporal analysis
495         heatmap_data = [[0 for _ in hours] for _ in days]
```

```
496
497         for email in self.emails:
498             day_idx = email.date.weekday()
499             hour = email.date.hour
500             heatmap_data[day_idx][hour] += 1
501
502         fig, ax = plt.subplots(1, 1, figsize=(16, 8))
503
504         im = ax.imshow(heatmap_data, cmap='YlOrRd', aspect='auto')
505
506         ax.set_xticks(range(len(hours)))
507         ax.set_yticks(range(len(days)))
508         ax.set_xticklabels(hours)
509         ax.set_yticklabels(days)
510
511         cbar = plt.colorbar(im, ax=ax)
512         cbar.set_label('Number of Emails', rotation=270, labelpad=20)
513
514         ax.set_title('Email Activity Heatmap (Day vs Hour)', fontsize=16, fontweight='bold', pad=20)
515         ax.set_xlabel('Hour of Day', fontsize=12)
516         ax.set_ylabel('Day of Week', fontsize=12)
517
518         plt.tight_layout()
519         plt.savefig(f"{self.output_dir}/activity_heatmap.png", dpi=300, bbox_inches='tight')
520         plt.close()
521
522     def _generate_network_analysis(self):
523         # WHY network analysis: Identifies communication patterns and potential threat vectors
524         connections = Counter()
525         for email in self.emails:
526             sender_domain = email.sender.split('@')[-1] if '@' in email.sender else email.sender
527             recipient_domain = email.recipient.split('@')[-1] if '@' in email.recipient else email.recipient
528             connections[(sender_domain, recipient_domain)] += 1
529
530         fig, ax = plt.subplots(1, 1, figsize=(12, 8))
531
532         top_connections = connections.most_common(10)
533         labels = [f"{sender} -> {recipient}" for (sender, recipient), count in top_connections]
534         counts = [count for (sender, recipient), count in top_connections]
535
536         bars = ax.barh(range(len(labels)), counts, color='lightcoral')
537         ax.set_yticks(range(len(labels)))
538         ax.set_yticklabels(labels)
539         ax.set_xlabel('Number of Emails')
540         ax.set_title('Top Email Communication Paths', fontsize=16, fontweight='bold', pad=20)
```

APPENDIX A 7/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
        # WHY value labels: Eliminates need for precise bar length comparison
        for i, bar in enumerate(bars):
            width = bar.get_width()
            ax.text(width + 0.1, bar.get_y() + bar.get_height()/2,
                    f'{int(width)}', ha='left', va='center')

        plt.tight_layout()
        plt.savefig(f"{self.output_dir}/network_analysis.png", dpi=300, bbox_inches='tight')
        plt.close()

    def _generate_severity_distribution(self):
        # WHY severity chart: Enables risk-based prioritization for response
        severity_counts = Counter(finding.severity for finding in self.findings)

        fig, ax = plt.subplots(1, 1, figsize=(10, 6))

        severities = ['Low', 'Medium', 'High']
        counts = [severity_counts.get(severity, 0) for severity in severities]
        colors = ['#2ecc71', '#f39c12', '#e74c3c']

        bars = ax.bar(severities, counts, color=colors, alpha=0.8)

        # WHY value labels: Critical for understanding absolute numbers
        for bar in bars:
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height + 0.1,
                    f'{int(height)}', ha='center', va='bottom', fontweight='bold')

        ax.set_title('Findings by Severity Level', fontsize=16, fontweight='bold', pad=20)
        ax.set_ylabel('Number of Findings', fontsize=12)
        ax.grid(True, alpha=0.3, axis='y')

        plt.tight_layout()
        plt.savefig(f"{self.output_dir}/severity_distribution.png", dpi=300, bbox_inches='tight')
        plt.close()

class ReportAgent:
    # WHY Report Agent: Consolidates findings into multiple formats for different stakeholders

    def __init__(self, emails: List[SimpleEmail], findings: List[Finding]):
        self.emails = emails
        self.findings = findings
        self.output_dir = "output/reports"
        os.makedirs(self.output_dir, exist_ok=True)
```

```
    def generate_comprehensive_report(self):
        # WHY dual format: Text for portability, HTML for rich visualization
        self._generate_text_report()
        self._generate_html_report()
        print("Report generation complete!")

    def _generate_text_report(self):
        # WHY text format: Portable, searchable, suitable for log integration
        stats = AnalysisAgent(self.emails).get_statistics()

        report_content = f"""
EMAIL FORENSICS ANALYSIS REPORT
==============================
Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}

EXECUTIVE SUMMARY
-----------------
Total Emails Analyzed: {stats['total_emails']}
Suspicious Emails: {stats['suspicious_emails']} ({stats['suspicious_emails']/stats['total_emails']*100:.1f}%)
External Communications: {stats['external_emails']}
After-Hours Communications: {stats['after_hours_emails']}
Total Security Findings: {stats['total_findings']}

FINDINGS BREAKDOWN
------------------
High Severity: {stats['high_severity_findings']}
Medium Severity: {stats['medium_severity_findings']}
Low Severity: {stats['low_severity_findings']}

DETAILED FINDINGS
-----------------
"""

        # WHY severity sorting: Prioritizes critical threats for immediate attention
        for finding in sorted(self.findings, key=lambda x: {'High': 3, 'Medium': 2, 'Low': 1}[x.severity], reverse=True):
            report_content += f"""
Finding: {finding.finding_type}
Severity: {finding.severity}
Email ID: {finding.email_id}
Description: {finding.description}
Timestamp: {finding.timestamp.strftime('%Y-%m-%d %H:%M:%S')}
{'='*50}
"""
```

APPENDIX A 8/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
        with open(f"{self.output_dir}/forensics_report.txt", 'w', encoding='utf-8') as f:
            f.write(report_content)

    def _generate_html_report(self):
        # WHY HTML format: Rich formatting, embedded visualizations, interactive elements
        stats = AnalysisAgent(self.emails).get_statistics()

        html_template = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Email Forensics Analysis Report</title>
    <style>
        body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; margin: 0; padding: 20px; background-color: #f5f5f5; }
        .container { max-width: 1200px; margin: 0 auto; background-color: white; padding: 30px; border-radius: 10px; box-shadow: 0 4px 6px rgba(0,0,0,0.1); }
        .header { text-align: center; margin-bottom: 30px; padding: 20px; background: linear-gradient(135deg, #667eea 0%, #764ba2 100%); color: white; border-radius: 8px; }
        .section { margin: 25px 0; }
        .finding {
            background-color: #f8f9fa;
            padding: 15px;
            margin: 15px 0;
            border-radius: 8px;
            border-left: 5px solid #ffc107;
            transition: transform 0.2s ease, box-shadow 0.2s ease;
        }
        .finding:hover { transform: translateY(-3px); box-shadow: 0 4px 12px rgba(0,0,0,0.1); }
        .high { border-left-color: #e74c3c; background-color: #fbeae5; }
        .medium { border-left-color: #f39c12; background-color: #fef5e7; }
        .low { border-left-color: #2ecc71; background-color: #eafaf1; }
        .stats {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
            gap: 20px;
            margin: 20px 0;
        }
        .stat-box {
            text-align: center;
            padding: 20px;
            background-color: #34495e;
            color: white;
            border-radius: 10px;
        }
```

```
        .stat-number { font-size: 2.2em; font-weight: 700; }
        .visualizations {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(400px, 1fr));
            gap: 20px;
            margin: 20px 0;
        }
        .viz-item { text-align: center; padding: 15px; background-color: #f8f9fa; border-radius: 8px; }
        .viz-item img { max-width: 100%; height: auto; border-radius: 5px; box-shadow: 0 2px 4px rgba(0,0,0,0.05); }
        footer { text-align: center; margin-top: 30px; color: #7f8c8d; font-size: 0.9em; }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>Email Forensics Analysis Report</h1>
            <p>Multi-Agent System & Advanced Analytics Dashboard</p>
        </div>

        <div class="section">
            <h2>Executive Summary</h2>
            <div class="stats">
                <div class="stat-box">
                    <div class="stat-number">{{ stats.total_emails }}</div>
                    <div>Total Emails</div>
                </div>
                <div class="stat-box">
                    <div class="stat-number">{{ stats.suspicious_emails }}</div>
                    <div>Suspicious Emails</div>
                </div>
                <div class="stat-box">
                    <div class="stat-number">{{ stats.total_findings }}</div>
                    <div>Security Findings</div>
                </div>
                <div class="stat-box">
                    <div class="stat-number">{{ stats.high_severity_findings }}</div>
                    <div>High Risk</div>
                </div>
            </div>
        </div>

        <div class="section">
            <h2>Key Findings</h2>
            {% for finding in findings %}
            <div class="finding {{ finding.severity.lower() }}">
```

APPENDIX A 9/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
                <h4>{{ finding.finding_type }} - {{ finding.severity }} Severity</h4>
                <p><strong>Email:</strong> {{ finding.email_id }}</p>
                <p><strong>Description:</strong> {{ finding.description }}</p>
                <p><strong>Detected:</strong> {{ finding.timestamp.strftime('%Y-%m-%d %H:%M:%S') }}</p>
            </div>
            {% endfor %}
        </div>

        <div class="section">
            <h2>Visualizations Dashboard</h2>
            <div class="visualizations">
                <div class="viz-item">
                    <h3>Summary Statistics</h3>
                    <img src="../visualizations/summary_chart.png" alt="Summary Chart">
                </div>
                <div class="viz-item">
                    <h3>Email Distribution</h3>
                    <img src="../visualizations/email_distribution_pie.png" alt="Distribution Pie Chart">
                </div>
                <div class="viz-item">
                    <h3>Hourly Activity</h3>
                    <img src="../visualizations/hourly_distribution.png" alt="Hourly Distribution">
                </div>
                <div class="viz-item">
                    <h3>Subject Analysis</h3>
                    <img src="../visualizations/wordcloud.png" alt="Word Cloud">
                </div>
                <div class="viz-item">
                    <h3>Timeline Analysis</h3>
                    <img src="../visualizations/timeline.png" alt="Timeline">
                </div>
                <div class="viz-item">
                    <h3>Activity Heatmap</h3>
                    <img src="../visualizations/activity_heatmap.png" alt="Activity Heatmap">
                </div>
                <div class="viz-item">
                    <h3>Communication Patterns</h3>
                    <img src="../visualizations/network_analysis.png" alt="Network Analysis">
                </div>
                <div class="viz-item">
                    <h3>Risk Assessment</h3>
                    <img src="../visualizations/severity_distribution.png" alt="Severity Distribution">
                </div>
            </div>
        </div>
```

```
        <footer>
            <p>Report generated by Multi-Agent Email Forensics System | {{ timestamp }}</p>
        </footer>
    </div>
</body>
</html>
        """

        template = Template(html_template)
        html_content = template.render(
            stats=stats,
            findings=self.findings,
            timestamp=datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        )

        with open(f"{self.output_dir}/forensics_report.html", 'w', encoding='utf-8') as f:
            f.write(html_content)

def generate_uml_documentation():
    # WHY UML diagrams: Provides visual documentation for system architecture and interactions
    uml_dir = "output/uml_documentation"
    os.makedirs(uml_dir, exist_ok=True)

    # WHY Mermaid format: Widely supported, GitHub-compatible, easy to render
    class_diagram_mermaid = '''classDiagram
    direction LR

    class DiscoveryAgent {
        -search_directory: str
        -discovered_files: List<str>
        +find_email_files(): List<str>
        +load_emails(): List<SimpleEmail>
    }

    class DashboardAgent {
        -emails: List<SimpleEmail>
        -findings: List<Finding>
        +generate_dashboard(): void
    }

    class AnalysisAgent {
        -emails: List<SimpleEmail>
        -findings: List<Finding>
        +analyze_emails(): List<Finding>
```

APPENDIX A 10/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
        +get_statistics(): Dict
    }

    class ReportAgent {
        -emails: List<SimpleEmail>
        -findings: List<Finding>
        +generate_comprehensive_report(): void
    }

    class SimpleEmail {
        +id: str
        +subject: str
        +sender: str
        +recipient: str
        +date: datetime
        +content: str
        +file_path: str
        +is_suspicious(): bool
        +is_after_hours(): bool
        +is_external(): bool
    }

    class Finding {
        +finding_type: str
        +description: str
        +email_id: str
        +severity: str
        +timestamp: datetime
    }

    DiscoveryAgent --|> SimpleEmail : "loads"
    DashboardAgent --|> SimpleEmail : "visualizes"
    AnalysisAgent --|> Finding : "creates"
    ReportAgent --|> Finding : "reports"
'''

    # WHY PlantUML format: Standard for sequence diagrams, tools support
    sequence_diagram = '''@startuml EmailForensicsSequence
participant "Main Controller" as Main
participant "EnhancedEmailGenerator" as Generator
participant "DiscoveryAgent" as Discovery
participant "AnalysisAgent" as Analysis
participant "DashboardAgent" as Dashboard
participant "ReportAgent" as Report

Main -> Generator: generate_emails(50, 30%)
Generator -> Main: emails[]
```

```

Main -> Discovery: find_email_files()
Discovery -> Discovery: load_emails()
Discovery -> Main: loaded_emails[]

Main -> Analysis: analyze_emails(loaded_emails)
Analysis -> Analysis: _keyword_analysis()
Analysis -> Analysis: _timing_analysis()
Analysis -> Analysis: _external_communication_analysis()
Analysis -> Main: findings[]

Main -> Dashboard: generate_dashboard()
Dashboard -> Dashboard: _generate_summary_chart()
Dashboard -> Dashboard: _generate_pie_chart()
Dashboard -> Dashboard: _generate_wordcloud()

Main -> Report: generate_comprehensive_report()
Report -> Report: _generate_text_report()
Report -> Report: _generate_html_report()
@enduml'''

    with open(f"{uml_dir}/class_diagram.md", "w", encoding="utf-8") as f:
        f.write(class_diagram_mermaid)

    with open(f"{uml_dir}/sequence_diagram.puml", "w", encoding="utf-8") as f:
        f.write(sequence_diagram)

    # WHY README: Provides viewing instructions for different platforms
    readme_content = """# UML Documentation

This directory contains UML diagrams for the Email Forensics System:

## Available Diagrams

### 1. Class Diagram (class_diagram.md) - Mermaid Format
- Shows the structure of all agent classes
- Illustrates relationships between components
- Displays class attributes and methods
- **Format**: Mermaid syntax

### 2. Sequence Diagram (sequence_diagram.puml) - PlantUML Format
- Shows the interaction flow between agents
- Illustrates the complete analysis process
- Displays method calls and data flow
- **Format**: PlantUML syntax
```

```

## Viewing the Diagrams

### For Mermaid Class Diagram:
- Online: https://mermaid.live/
- VS Code: Mermaid Preview extension
- GitHub: Native Mermaid support in README files
- Obsidian: Native Mermaid support

### For PlantUML Sequence Diagram:
- Online: https://www.planttext.com/
- VS Code: PlantUML extension
- IntelliJ IDEA: PlantUML integration plugin

Simply copy and paste the diagram content into any compatible viewer.
"""

    with open(f"{uml_dir}/README.md", "w", encoding="utf-8") as f:
        f.write(readme_content)

    print("UML documentation generated successfully!")
    print(f"- Class diagram (Mermaid): {uml_dir}/class_diagram.md")
    print(f"- Sequence diagram (PlantUML): {uml_dir}/sequence_diagram.puml")
    print(f"- Documentation: {uml_dir}/README.md")

# WHY main execution function: Orchestrates complete workflow with clear logging
def run_email_forensics_system():
    """Run the complete email forensics analysis system"""
    print("Starting Multi-Agent Email Forensics System")
    print("=" * 50)

    # WHY Step 1: Generate synthetic data to avoid ethical issues with real phishing emails
    print("Generating test emails")
    generator = EnhancedEmailGenerator()
    emails = generator.generate_emails(50, 0.3)

    # WHY Step 2: Test file-based discovery to simulate real forensic scenarios
    print("Discovering email files")
    discovery_agent = DiscoveryAgent()
    discovered_files = discovery_agent.find_email_files()
    loaded_emails = discovery_agent.load_emails()

    # WHY Step 3: Multi-faceted analysis for comprehensive threat detection
    print("Analyzing emails for security threats")
    analysis_agent = AnalysisAgent(loaded_emails)
    findings = analysis_agent.analyze_emails()
    stats = analysis_agent.get_statistics()
```

APPENDIX A 11/11

Integrated Multi-Agent Email Forensics Notebook - Final Version

```
    # WHY Step 4: Visual analytics for pattern recognition and reporting
    print("Generating comprehensive dashboard")
    dashboard_agent = DashboardAgent(loaded_emails, findings)
    dashboard_agent.generate_dashboard()

    # WHY Step 5: Dual-format reports for different stakeholder needs
    print("Generating detailed reports")
    report_agent = ReportAgent(loaded_emails, findings)
    report_agent.generate_comprehensive_report()

    # WHY Step 6: Visual documentation for system understanding
    print("Generating UML documentation")
    generate_uml_documentation()

    # WHY summary output: Confirms successful execution and guides user to outputs
    print("\n" + "=" * 50)
    print("EMAIL FORENSICS ANALYSIS COMPLETE!")
    print("=" * 50)
    print(f"Statistics:")
    print(f"   • Total emails processed: {stats['total_emails']}")
    print(f"   • Suspicious emails detected: {stats['suspicious_emails']}")
    print(f"   • Security findings: {stats['total_findings']}")
    print(f"   • High-risk findings: {stats['high_severity_findings']}")

    print(f"Generated files:")
    print(f"   • Emails: output/emails/")
    print(f"   • Visualizations: output/visualizations/")
    print(f"   • Reports: output/reports/")
    print(f"   • UML Documentation: output/uml_documentation/")

    print(f"Key outputs:")
    print(f"   • HTML Report: output/reports/forensics_report.html")
    print(f"   • Class Diagram: output/uml_documentation/class_diagram.md")
    print(f"   • Sequence Diagram: output/uml_documentation/sequence_diagram.puml")

    return {
        'emails': loaded_emails,
        'findings': findings,
        'statistics': stats
    }

# WHY __main__ guard: Prevents execution when imported as module
if __name__ == "__main__":
    results = run_email_forensics_system()
```

Trevisi et al. (2025)

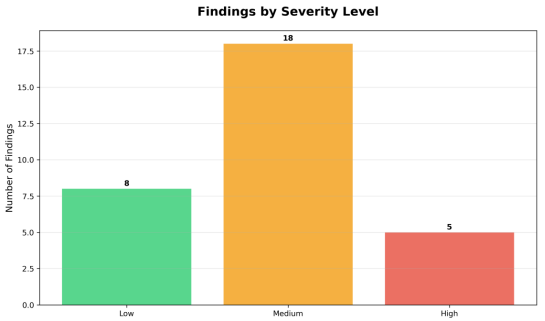
APPENDIX B

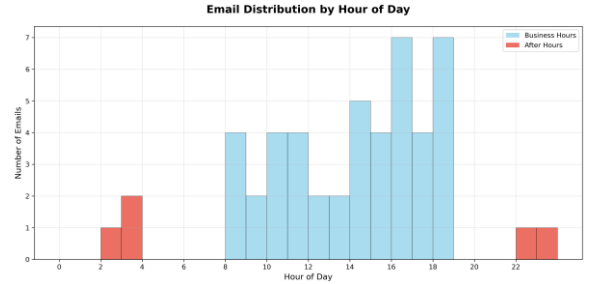
Simple test file to demonstrate testing works

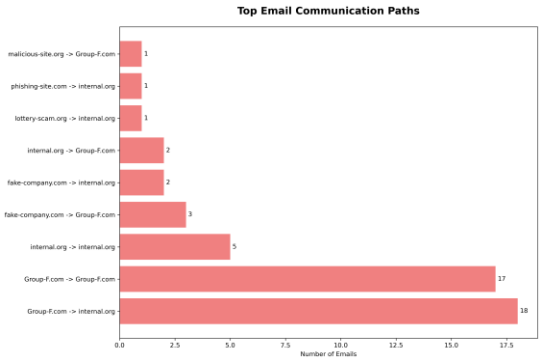
```python
"""
Simple test file to demonstrate testing works
Tests only the SimpleEmail class
"""

import pytest
from datetime import datetime
from dataclasses import dataclass


#SimpleEmail class here for testing
@dataclass
class SimpleEmail:
    id: str
    subject: str
    sender: str
    recipient: str
    date: datetime
    content: str
    file_path: str

    def is_suspicious(self) -> bool:
        suspicious_words = [
            'confidential', 'secret', 'critical', 'account', 'payment', 'transfer',
            'urgent', 'immediate', 'verify', 'suspend', 'click here', 'download',
            'invoice', 'refund', 'winner', 'congratulations', 'inheritance',
            'million', 'dollars', 'bitcoin', 'cryptocurrency', 'phishing'
        ]
        text_to_check = (self.subject + " " + self.content).lower()
        return any(word in text_to_check for word in suspicious_words)

    def is_after_hours(self) -> bool:
        return self.date.hour < 8 or self.date.hour > 18

    def is_external(self) -> bool:
        internal_domains = ['Group-F.com', 'internal.org']
        sender_domain = self.sender.split('@')[-1] if '@' in self.sender else ''
        return sender_domain not in internal_domains


# TESTS START HERE

def test_suspicious_email_with_urgent():
    """Test that email with 'urgent' is flagged as suspicious"""
    email = SimpleEmail(
        id="test_001",
        subject="URGENT: Verify your account",
        sender="phishing@bad.com",
        recipient="user@Group-F.com",
        date=datetime(2025, 1, 15, 10, 0),
        content="Please verify immediately",
        file_path="test.txt"
    )

    assert email.is_suspicious() == True
    print("✓ Suspicious email detected correctly")


def test_normal_email_not_suspicious():
    """Test that normal email is NOT flagged"""
    email = SimpleEmail(
        id="test_002",
        subject="Weekly meeting agenda",
        sender="john@Group-F.com",
        recipient="team@Group-F.com",
        date=datetime(2025, 1, 15, 10, 0),
        content="Please find the agenda attached",
        file_path="test.txt"
    )

    assert email.is_suspicious() == False
    print("✓ Normal email passed correctly")


def test_after_hours_detection():
    """Test that email at 2 AM is flagged as after-hours"""
    email = SimpleEmail(
        id="test_003",
        subject="Late night email",
        sender="user@Group-F.com",
        recipient="user2@Group-F.com",
        date=datetime(2025, 1, 15, 2, 30),  # 2:30 AM
        content="Working late",
        file_path="test.txt"
    )

    assert email.is_after_hours() == True
    print("✓ After-hours email detected correctly")


def test_business_hours_not_flagged():
    """Test that email at 10 AM is NOT flagged as after-hours"""
    email = SimpleEmail(
        recipient="user2@Group-F.com",
        date=datetime(2025, 1, 15, 10, 0),  # 10:00 AM
        content="Good morning",
        file_path="test.txt"
    )

    assert email.is_after_hours() == False
    print("✓ Business hours email passed correctly")


def test_external_domain_detection():
    """Test that external email is flagged"""
    email = SimpleEmail(
        id="test_005",
        subject="External email",
        sender="contact@external-company.com",
        recipient="sales@Group-F.com",
        date=datetime(2025, 1, 15, 14, 0),
        content="Business inquiry",
        file_path="test.txt"
    )

    assert email.is_external() == True
    print("✓ External email detected correctly")


def test_internal_domain_not_external():
    """Test that internal Group-F email is NOT flagged as external"""
    email = SimpleEmail(
        id="test_006",
        subject="Internal email",
        sender="employee@Group-F.com",
        recipient="manager@Group-F.com",
        date=datetime(2025, 1, 15, 14, 0),
        content="Internal communication",
        file_path="test.txt"
    )

    assert email.is_external() == False
    print("✓ Internal email passed correctly")


if __name__ == "__main__":
    # Run tests when file is executed directly
    pytest.main([__file__, "-v", "-s"])
```

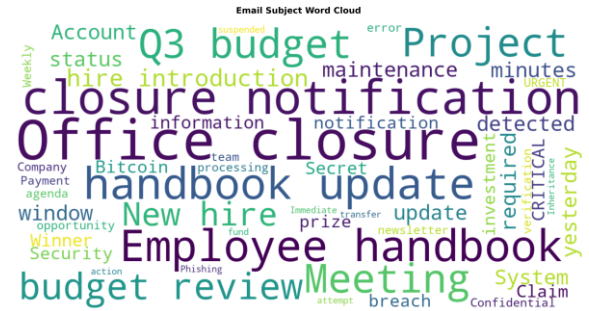
APPENDIX C

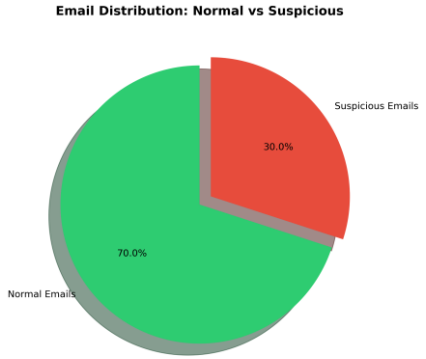
Visualizations

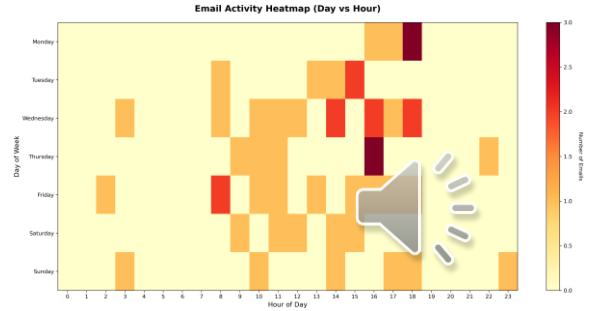






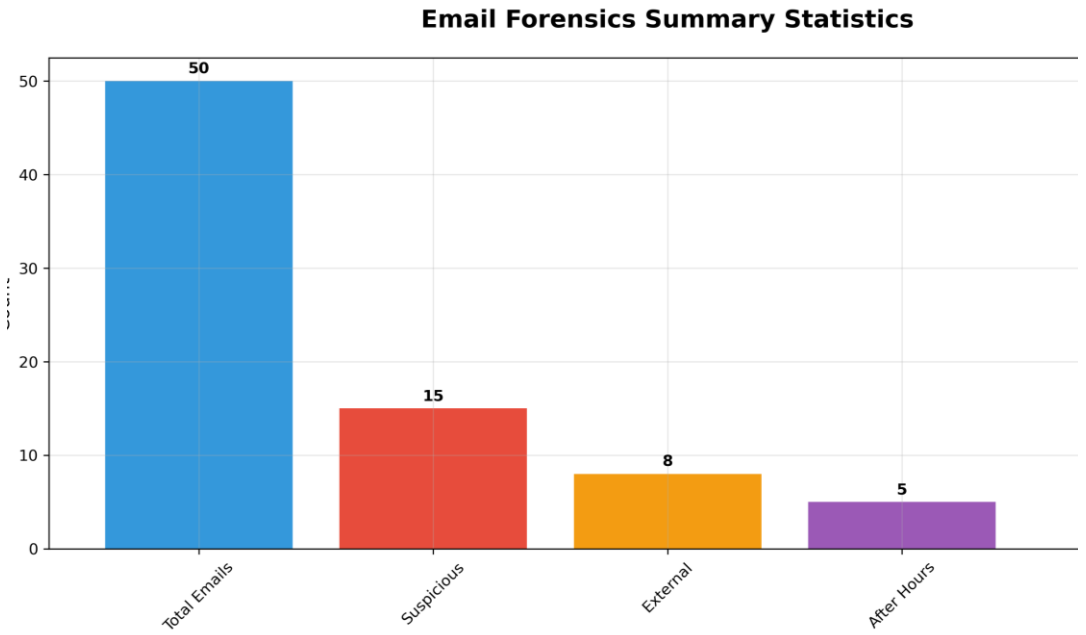


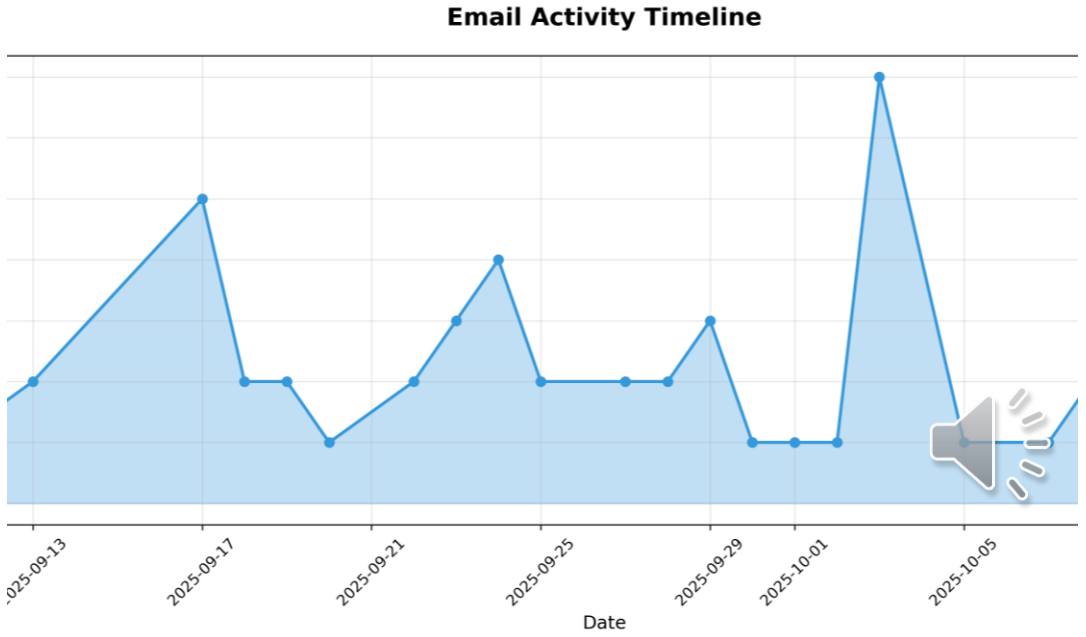




APPENDIX C

Visualizations





Thank you for the attention

INTELLIGENT AGENT SYSTEM FOR SECURITY THREAT DETECTION

STUDENT ID: 219155