בדאטה יש את העמודות הבאות: employee\_id date employee\_department employee\_campus employee\_job employee\_seniority\_years is\_contractor employee\_classification has\_foreign\_citizenship has\_criminal\_record has\_medical\_history employee\_origin\_country is\_malicious risk\_travel\_indicator num\_print\_commands total\_printed\_pages num\_print\_commands\_off\_hours num\_printed\_pages\_off\_hours num\_color\_prints num\_bw\_prints ratio\_color\_prints printed\_from\_other print\_campuses num\_burn\_requests max\_request\_classification avg\_request\_classification num\_burn\_requests\_off\_hours total\_burn\_volume\_mb total\_files\_burned burned\_from\_other burn\_campuses is\_abroad trip\_day\_number country\_name is\_hostile\_country\_trip is\_official\_trip is\_origin\_country\_trip num\_entries num\_exits first\_entry\_time last\_exit\_time total\_presence\_minutes entered\_during\_night\_hours num\_unique\_campus early\_entry\_flag late\_exit\_flag entry\_during\_weekend

שמות המחלקות/התפקידים אמורים להיות כפי שמות המחלקות והתפקידים להלן, ורק באנגלית:

Department Department (Hebrew) Position Position (Hebrew)

Executive Management הנהלה בכירה Chief Executive Officer (CEO) "מנכ""ל"

Executive Management הנהלה בכירה Chief Legal Officer "סמנכ""ל משפטי"

Executive Management הנהלה בכירה Chief Human Resources Officer (CHRO) "סמנכ""ל משאבי אנוש"

Executive Management הנהלה בכירה Chief Information Officer (CIO) "סמנכ""ל מערכות מידע"

Executive Management הנהלה בכירה Chief Technology Officer (CTO) "סמנכ""ל טכנולוגיות"

Executive Management הנהלה בכירה Chief Operating Officer (COO) "סמנכ""ל תפעול"

Executive Management הנהלה בכירה Chief Financial Officer (CFO) "סמנכ""ל כספים"

Executive Management הנהלה בכירה Chief Marketing and Business Development Officer "סמנכ""ל שיווק ופיתוח עסקי"

Executive Management הנהלה בכירה Secretary מזכיר/ה

R&D Department "מחלקת מו""פ" Head of R&D "ראש תחום מו""פ"

R&D Department "מחלקת מו""פ" Systems Engineer מהנדס מערכת

R&D Department "מחלקת מו""פ" Development Engineer (Hardware / Software / Mechanical) מהנדס פיתוח (חומרה / תוכנה / מכניקה)

R&D Department "מחלקת מו""פ" Algorithm Engineer מהנדס אלגוריתמים

R&D Department "מחלקת מו""פ" Integration and Testing Engineer מהנדס אינטגרציה ובדיקות

R&D Department "מחלקת מו""פ" Secretary מזכיר/ה

Engineering Department מחלקת הנדסה Process Engineer מהנדס תהליך

Engineering Department מחלקת הנדסה Design Engineer מהנדס תכן

Engineering Department מחלקת הנדסה Head of Engineering ראש תחום הנדסה

Engineering Department מחלקת הנדסה Systems Engineer מהנדס מערכת

Engineering Department מחלקת הנדסה Test Engineer מהנדס בדיקות

Engineering Department מחלקת הנדסה Secretary מזכיר/ה

Operations and Manufacturing מחלקת תפעול וייצור Operations Manager מנהל תפעול

Operations and Manufacturing מחלקת תפעול וייצור Manufacturing Engineer מהנדס ייצור

Operations and Manufacturing מחלקת תפעול וייצור Logistics Manager מנהל לוגיסטיקה

Operations and Manufacturing מחלקת תפעול וייצור Procurement Officer אחראי רכש

Operations and Manufacturing מחלקת תפעול וייצור Warehouse Manager מנהל מחסן

Operations and Manufacturing מחלקת תפעול וייצור Secretary מזכיר/ה

Project Management ניהול פרויקטים Project Manager מנהל פרויקטים

Project Management ניהול פרויקטים Project Engineer מהנדס פרויקט

Project Management ניהול פרויקטים Project Coordinator מתאם פרויקטים

Project Management ניהול פרויקטים Secretary מזכיר/ה

Security and Information Security אבטחה ואבטחת מידע Physical Access Control בקרת כניסה וביטחון פיזי

Security and Information Security אבטחה ואבטחת מידע Information Security Investigator חוקר אבטחת מידע

Security and Information Security אבטחה ואבטחת מידע Cyber Analyst אנליסט סייבר

Security and Information Security אבטחה ואבטחת מידע Chief Information Security Officer (CISO) מנהל אבטחת מידע (CISO)

Security and Information Security אבטחה ואבטחת מידע Security Officer קצין ביטחון

Human Resources משאבי אנוש HR Manager מנהל/ת משאבי אנוש

Human Resources משאבי אנוש Recruitment Coordinator רכז/ת גיוס

Human Resources משאבי אנוש Employee Welfare Coordinator רכז/ת רווחה

Human Resources משאבי אנוש Training Coordinator אחראי הדרכה

Human Resources משאבי אנוש Secretary מזכיר/ה

Legal and Regulation משפטים ורגולציה Regulatory Affairs Officer אחראי רגולציה

Legal and Regulation משפטים ורגולציה Defense Export Compliance Officer אחראי יצוא ביטחוני

Legal and Regulation משפטים ורגולציה Legal Advisor יועץ משפטי

Finance מחלקת כספים Accountant חשב

Finance מחלקת כספים Financial Analyst אנליסט פיננסי

Finance מחלקת כספים Budget Manager מנהל תקציב

Finance מחלקת כספים Finance Manager מנהל כספים

Finance מחלקת כספים Secretary מזכיר/ה

Marketing and Business Development שיווק ופיתוח עסקי Business Development Manager מנהל פיתוח עסקי

Marketing and Business Development שיווק ופיתוח עסקי Account Manager מנהל לקוחות

Marketing and Business Development שיווק ופיתוח עסקי Bid Coordinator מתאם מכרזים

Marketing and Business Development שיווק ופיתוח עסקי Marketing Manager מנהל שיווק

Marketing and Business Development שיווק ופיתוח עסקי Secretary מזכיר/ה

Information Technology מחלקת מערכות מידע IT Director מנהל מערכות מידע

Information Technology מחלקת מערכות מידע Information Security Specialist מומחה אבטחת מידע

Information Technology מחלקת מערכות מידע Systems and Network Administrator איש סיסטם ורשתות

Information Technology מחלקת מערכות מידע BI Developer / Data Analyst מפתח BI / דאטה אנליסט

Information Technology מחלקת מערכות מידע Enterprise Systems Developer (ERP / CRM / SAP) מפתח מערכות פנים ארגוניות (ERP / CRM / SAP)

Information Technology מחלקת מערכות מידע Data Scientist מדען נתונים

Information Technology מחלקת מערכות מידע Secretary מזכיר/ה

על מנת שהדאטה שלנו יהיה ככל האפשר קרוב למציאות, עליך לבצע דפוסי התנהגות שונים לבעלי תפקידים שונים, כך שלכל תפקיד תהיה התנהגות קבוצתית דומה השונה מהתנהגויות הקבוצות האחרות (לדוגמה, זה טבעי שמזכירות מדפיסות יותר ממתכנתים, אך כמובן שזהו אינו ההבדל היחיד בין המזכירות למתכנתים וההדפסות הוא רק סממן אחד מההתנהגות האופיינית למזכירות. דוגמה נוספת: לעובדי שיווק יהיו יותר הדפסות בצבע מאשר מתכנתים, ואם נניח הדפסה בצבע למתכנת היא חשודה היות שבדרך כלל אינה נצרכת לתוקף עבודה – לבעל תפקיד שמתוקף תפקידו עליו להדפיס בצבע – הדפסת כמות עמודי צבע בכמות שנחשבת סבירה לתפקיד זה - לא תהיה מחשידה).

חשוב מהם הדפוסים האפשריים לגבי הדאטה והעמודות בו והשתדל להיצמד ככל האפשר לדפוסים מציאותיים. לפני שתיצור את הדאטה אמור לנו מהם הדפוסים שחשבת לעשות ואנחנו נאמר לך אם אנחנו מאשרים זאת.

יש לציין שעובדים בעלי אופי תפקיד דומה יהיו בעלי התנהגות דומה מבחינת זמני הגעה ויציאה, דפוסי הדפסה, צריבה, נסיעות לחול בתפקיד וכו'.

כאשר עובדים זדוניים צריכים לחרוג ביחס למקובל בתפקיד שלהם, או ביחס לעצמם לאורך ציר הזמן או ביחס לשניהם.

כלומר שמור על כל מה שהגדרת עד היום, רק שתוסיף שונות בין תפקידים שונים ודמיון בתוך תפקידים דומים. אין צורך ליצור שונות בין כל תפקיד ותפקיד. אלא בהתאם לאופי העבודה המקובל בשוק- למשל כלל המפתחים בד"כ עובדים לפי שעון גמיש, לעומת מחלקות שצריכות לתת שירות כמו מוקדנים שעובדים בשעות קבועות יותר.

בנוסף יהיה טוב שתפרט מה היו הנחות היסוד שלך ואלו תפקידים הגדרת כבעלי דפוס עבודה דומה.

בנוסף: בעבר כתבתי לך "אף עובד אינו מועסק באופן רשמי ביום שישי לאחר 4 אחרי הצהריים או בשבת" ולפי זה יצרת את הדאטה. אבל זה לא מדויק כי אנשי אבטחה מאבטחים את המקום 24/7 וחלק מהעבודה שלנו תהיה להבחין שעבור עובדי אבטחה זו התנהגות תקינה אז כן צריך לייצר מקרים כאלו בתפקיד זה.

אני מצרפת לך את הקוד שיצר את הדאטה בעבר:

self.employee\_trips = {} *# {emp\_id: {'country': str, 'start\_date': date, 'duration': int, 'is\_official': bool}}*

def \_generate\_employee\_profiles(self):

"""Generate realistic employee profiles"""

employees = {}

num\_digits = len(str(self.num\_employees))

for i in range(self.num\_employees):

emp\_id = str(i + 1).zfill(num\_digits) *# Ensure same number of digits*

*# Assign origin country*

origin\_country = np.random.choice(self.origin\_countries, p=[0.05, 0.03, 0.02, 0.05, 0.05, 0.2, 0.15, 0.1, 0.1, 0.1, 0.15])

employees[emp\_id] = {

'department': np.random.choice(self.departments),

'campus': np.random.choice(self.campuses),

'job': np.random.choice(self.jobs),

'seniority\_years': np.random.randint(0, 51), *# 0 to 50 years*

'is\_contractor': np.random.choice([0, 1], p=[0.8, 0.2]),

'classification': np.random.choice([1, 2, 3, 4], p=[0.4, 0.3, 0.2, 0.1]),

'foreign\_citizenship': np.random.choice([0, 1], p=[0.85, 0.15]),

'criminal\_record': np.random.choice([0, 1], p=[0.95, 0.05]),

'medical\_history': np.random.choice([0, 1], p=[0.85, 0.15]),

'origin\_country': origin\_country

}

return employees

def \_is\_off\_hours(self, dt):

"""Check if datetime is during off hours"""

if dt.weekday() == 4 and dt.hour >= 16: *# Friday after 16:00*

return True

if dt.weekday() == 5: *# Saturday*

return True

if dt.time() < datetime.strptime("05:00", "%H:%M").time() or dt.time() > datetime.strptime("21:00", "%H:%M").time():

return True

return False

def \_is\_weekend(self, dt):

"""Check if datetime is weekend"""

return dt.weekday() >= 4 *# Friday or Saturday*

def \_is\_night\_hours(self, dt):

"""Check if datetime is during night hours (22:00-05:00)"""

return dt.hour >= 22 or dt.hour <= 5

def \_generate\_print\_activity(self, emp\_id, date, is\_malicious, is\_abroad):

"""Generate realistic printing activity"""

base\_date = datetime.combine(date, datetime.min.time())

*# If employee is abroad and not malicious, very low chance of printing*

if is\_abroad and not is\_malicious:

if np.random.random() < 0.98: *# 98% chance of no printing when abroad*

return self.\_empty\_print\_activity()

*# If employee is abroad and malicious, moderate chance of printing (credential theft/collusion)*

if is\_abroad and is\_malicious:

if np.random.random() < 0.85: *# 85% chance of no printing when abroad (they're careful)*

return self.\_empty\_print\_activity()

*# Normal employees print less*

if not is\_malicious:

if np.random.random() < 0.7: *# 70% chance of no printing*

return self.\_empty\_print\_activity()

*# Normal printing patterns*

num\_commands = np.random.poisson(3)

total\_pages = sum(np.random.poisson(5) for \_ in range(num\_commands))

color\_ratio = np.random.beta(2, 8) *# Most prints are B&W*

else:

*# Malicious employees might print more, especially off-hours*

if np.random.random() < 0.9: *# Malicious employees print more often*

num\_commands = np.random.poisson(8)

total\_pages = sum(np.random.poisson(12) for \_ in range(num\_commands))

color\_ratio = np.random.beta(3, 5) *# More color prints (documents, charts)*

else:

return self.\_empty\_print\_activity()

*# Determine printing campuses*

employee\_campus = self.employees[emp\_id]['campus']

print\_campuses = set()

*# Decide how many campuses to print from*

if is\_malicious and np.random.random() < 0.3:

*# Malicious might print from multiple campuses*

num\_print\_campuses = np.random.choice([1, 2, 3], p=[0.5, 0.35, 0.15])

else:

*# Normal employees usually print from their own campus*

num\_print\_campuses = np.random.choice([1, 2], p=[0.9, 0.1])

*# Select campuses*

if num\_print\_campuses == 1:

if np.random.random() < 0.9: *# 90% chance own campus*

print\_campuses.add(employee\_campus)

else:

print\_campuses.add(np.random.choice(self.campuses))

else:

print\_campuses.add(employee\_campus) *# Always include own campus*

other\_campuses = [c for c in self.campuses if c != employee\_campus]

additional\_campuses = np.random.choice(other\_campuses, size=min(num\_print\_campuses-1, len(other\_campuses)), replace=False)

print\_campuses.update(additional\_campuses)

*# Check if printed from other campus*

printed\_from\_other = 1 if any(campus != employee\_campus for campus in print\_campuses) else 0

*# Calculate off-hours printing*

off\_hours\_commands = 0

off\_hours\_pages = 0

if is\_malicious and np.random.random() < 0.4: *# Malicious more likely to print off-hours*

off\_hours\_commands = max(0, int(num\_commands \* np.random.uniform(0.2, 0.8)))

off\_hours\_pages = max(0, int(total\_pages \* np.random.uniform(0.2, 0.8)))

elif np.random.random() < 0.1: *# Normal employees occasionally print off-hours*

off\_hours\_commands = max(0, int(num\_commands \* np.random.uniform(0.1, 0.3)))

off\_hours\_pages = max(0, int(total\_pages \* np.random.uniform(0.1, 0.3)))

num\_color = int(total\_pages \* color\_ratio)

num\_bw = total\_pages - num\_color

return {

'num\_print\_commands': num\_commands,

'total\_printed\_pages': total\_pages,

'num\_print\_commands\_off\_hours': off\_hours\_commands,

'num\_printed\_pages\_off\_hours': off\_hours\_pages,

'num\_color\_prints': num\_color,

'num\_bw\_prints': num\_bw,

'ratio\_color\_prints': color\_ratio if total\_pages > 0 else 0,

'printed\_from\_other': printed\_from\_other,

'print\_campuses': len(print\_campuses)

}

def \_empty\_print\_activity(self):

"""Return empty print activity"""

return {

'num\_print\_commands': 0,

'total\_printed\_pages': 0,

'num\_print\_commands\_off\_hours': 0,

'num\_printed\_pages\_off\_hours': 0,

'num\_color\_prints': 0,

'num\_bw\_prints': 0,

'ratio\_color\_prints': 0,

'printed\_from\_other': 0,

'print\_campuses': 0

}

def \_generate\_burn\_activity(self, emp\_id, date, is\_malicious, is\_abroad):

"""Generate CD/USB burning activity"""

*# If employee is abroad and not malicious, very low chance of burning*

if is\_abroad and not is\_malicious:

if np.random.random() < 0.99: *# 99% chance of no burning when abroad*

return self.\_empty\_burn\_activity()

*# If employee is abroad and malicious, low chance of burning (they're careful)*

if is\_abroad and is\_malicious:

if np.random.random() < 0.90: *# 90% chance of no burning when abroad*

return self.\_empty\_burn\_activity()

*# Most days have no burning activity*

if not is\_malicious and np.random.random() < 0.95:

return self.\_empty\_burn\_activity()

if is\_malicious and np.random.random() < 0.85:

return self.\_empty\_burn\_activity()

employee\_campus = self.employees[emp\_id]['campus']

employee\_classification = self.employees[emp\_id]['classification']

if is\_malicious:

num\_requests = np.random.poisson(3) + 1

*# Malicious users try to access higher classification data*

max\_classification = min(4, employee\_classification + np.random.choice([0, 1, 2], p=[0.3, 0.4, 0.3]))

classifications = [np.random.randint(1, max\_classification + 1) for \_ in range(num\_requests)]

volume\_mb = np.random.lognormal(8, 1.5) *# Larger volumes*

num\_files = np.random.poisson(50) + 10

else:

num\_requests = 1

max\_classification = min(employee\_classification, np.random.choice([1, 2, 3], p=[0.6, 0.3, 0.1]))

classifications = [max\_classification]

volume\_mb = np.random.lognormal(6, 1) *# Smaller volumes*

num\_files = np.random.poisson(10) + 1

*# Determine burning campuses*

burn\_campuses = set()

*# Decide how many campuses to burn from*

if is\_malicious and np.random.random() < 0.25:

num\_burn\_campuses = np.random.choice([1, 2, 3], p=[0.6, 0.3, 0.1])

else:

num\_burn\_campuses = 1

*# Select campuses*

if num\_burn\_campuses == 1:

if np.random.random() < 0.85: *# 85% chance own campus*

burn\_campuses.add(employee\_campus)

else:

burn\_campuses.add(np.random.choice(self.campuses))

else:

burn\_campuses.add(employee\_campus)

other\_campuses = [c for c in self.campuses if c != employee\_campus]

additional\_campuses = np.random.choice(other\_campuses, size=min(num\_burn\_campuses-1, len(other\_campuses)), replace=False)

burn\_campuses.update(additional\_campuses)

*# Check if burned from other campus*

burned\_from\_other = 1 if any(campus != employee\_campus for campus in burn\_campuses) else 0

*# Off-hours burning (more suspicious)*

off\_hours\_requests = 0

if is\_malicious and np.random.random() < 0.6:

off\_hours\_requests = max(0, int(num\_requests \* np.random.uniform(0.3, 0.8)))

elif np.random.random() < 0.05:

off\_hours\_requests = max(0, int(num\_requests \* 0.2))

return {

'num\_burn\_requests': num\_requests,

'max\_request\_classification': max(classifications),

'avg\_request\_classification': np.mean(classifications),

'num\_burn\_requests\_off\_hours': off\_hours\_requests,

'total\_burn\_volume\_mb': int(volume\_mb),

'total\_files\_burned': num\_files,

'burned\_from\_other': burned\_from\_other,

'burn\_campuses': len(burn\_campuses)

}

def \_empty\_burn\_activity(self):

"""Return empty burn activity"""

return {

'num\_burn\_requests': 0,

'max\_request\_classification': 0,

'avg\_request\_classification': 0,

'num\_burn\_requests\_off\_hours': 0,

'total\_burn\_volume\_mb': 0,

'total\_files\_burned': 0,

'burned\_from\_other': 0,

'burn\_campuses': 0

}

def \_update\_trip\_status(self, emp\_id, date):

"""Update trip status for employee"""

*# Check if employee has ongoing trip*

if emp\_id in self.employee\_trips:

trip = self.employee\_trips[emp\_id]

days\_since\_start = (date - trip['start\_date']).days

if days\_since\_start < trip['duration']:

*# Still on trip*

return {

'is\_abroad': 1,

'trip\_day\_number': days\_since\_start + 1,

'country\_name': trip['country'],

'is\_hostile\_country\_trip': 1 if trip['country'] in self.hostile\_countries else 0,

'is\_official\_trip': trip['is\_official'],

'is\_origin\_country\_trip': 1 if trip['country'] == self.employees[emp\_id]['origin\_country'] else 0

}

else:

*# Trip ended*

del self.employee\_trips[emp\_id]

return self.\_no\_travel\_activity()

*# Check if starting new trip*

if np.random.random() < 0.005: *# 0.5% chance of starting trip each day*

return self.\_start\_new\_trip(emp\_id, date)

return self.\_no\_travel\_activity()

def \_start\_new\_trip(self, emp\_id, date):

"""Start new trip for employee"""

is\_malicious = emp\_id in self.malicious\_employee\_ids

*# Select destination*

if is\_malicious and np.random.random() < 0.3:

*# Malicious more likely to visit hostile countries*

country = np.random.choice(self.hostile\_countries)

else:

*# Normal distribution of countries*

country = np.random.choice(self.countries)

*# Determine if official trip*

is\_official = np.random.choice([0, 1], p=[0.3, 0.7]) *# Most trips are official*

*# Check if visiting origin country*

origin\_country = self.employees[emp\_id]['origin\_country']

is\_origin\_trip = 1 if country == origin\_country else 0

*# If visiting origin country, more likely to be personal*

if is\_origin\_trip and np.random.random() < 0.6:

is\_official = 0

*# Trip duration*

duration = np.random.randint(1, 15) *# 1-14 days*

*# Store trip info*

self.employee\_trips[emp\_id] = {

'country': country,

'start\_date': date,

'duration': duration,

'is\_official': is\_official

}

return {

'is\_abroad': 1,

'trip\_day\_number': 1,

'country\_name': country,

'is\_hostile\_country\_trip': 1 if country in self.hostile\_countries else 0,

'is\_official\_trip': is\_official,

'is\_origin\_country\_trip': is\_origin\_trip

}

def \_no\_travel\_activity(self):

"""Return no travel activity"""

return {

'is\_abroad': 0,

'trip\_day\_number': None,

'country\_name': None,

'is\_hostile\_country\_trip': 0,

'is\_official\_trip': 0,

'is\_origin\_country\_trip': 0

}

def \_generate\_access\_activity(self, emp\_id, date, is\_malicious, is\_abroad):

"""Generate building access activity"""

*# If employee is abroad, very low chance of access (suspicious if happens)*

if is\_abroad:

if is\_malicious and np.random.random() < 0.05: *# 5% chance - credential theft*

pass *# Generate suspicious access*

elif not is\_malicious and np.random.random() < 0.001: *# 0.1% chance - error/family access*

pass *# Generate rare access*

else:

return self.\_empty\_access\_activity()

*# Normal absence chance*

if np.random.random() < 0.05: *# 5% chance of no access (sick day, vacation)*

return self.\_empty\_access\_activity()

base\_date = datetime.combine(date, datetime.min.time())

*# Normal work patterns*

if not is\_malicious:

num\_entries = np.random.choice([1, 2, 3], p=[0.7, 0.25, 0.05])

first\_entry = base\_date + timedelta(hours=np.random.normal(8, 1)) *# Around 8 AM*

last\_exit = base\_date + timedelta(hours=np.random.normal(17, 1.5)) *# Around 5 PM*

else:

*# Malicious employees might have irregular hours*

num\_entries = np.random.choice([1, 2, 3, 4], p=[0.5, 0.3, 0.15, 0.05])

if np.random.random() < 0.3: *# Sometimes very early or very late*

first\_entry = base\_date + timedelta(hours=np.random.uniform(5, 7))

last\_exit = base\_date + timedelta(hours=np.random.uniform(19, 23))

else:

first\_entry = base\_date + timedelta(hours=np.random.normal(8, 1.5))

last\_exit = base\_date + timedelta(hours=np.random.normal(17, 2))

*# Ensure logical order*

if last\_exit <= first\_entry:

last\_exit = first\_entry + timedelta(hours=np.random.uniform(4, 10))

num\_exits = num\_entries *# Assume balanced entries/exits*

*# Calculate presence time*

total\_minutes = int((last\_exit - first\_entry).total\_seconds() / 60)

*# Flags*

early\_entry = 1 if first\_entry.hour < 6 else 0

late\_exit = 1 if last\_exit.hour > 22 else 0

night\_entry = 1 if self.\_is\_night\_hours(first\_entry) else 0

weekend\_entry = 1 if self.\_is\_weekend(first\_entry) else 0

*# Multiple campus access (suspicious for malicious)*

employee\_campus = self.employees[emp\_id]['campus']

if is\_malicious and np.random.random() < 0.2:

num\_unique\_campus = np.random.choice([2, 3])

else:

num\_unique\_campus = 1

return {

'num\_entries': num\_entries,

'num\_exits': num\_exits,

'first\_entry\_time': first\_entry.strftime('%H:%M'),

'last\_exit\_time': last\_exit.strftime('%H:%M'),

'total\_presence\_minutes': total\_minutes,

'entered\_during\_night\_hours': night\_entry,

'num\_unique\_campus': num\_unique\_campus,

'early\_entry\_flag': early\_entry,

'late\_exit\_flag': late\_exit,

'entry\_during\_weekend': weekend\_entry

}

def \_empty\_access\_activity(self):

"""Return empty access activity"""

return {

'num\_entries': 0,

'num\_exits': 0,

'first\_entry\_time': None,

'last\_exit\_time': None,

'total\_presence\_minutes': 0,

'entered\_during\_night\_hours': 0,

'num\_unique\_campus': 0,

'early\_entry\_flag': 0,

'late\_exit\_flag': 0,

'entry\_during\_weekend': 0

}

def generate\_dataset(self):

"""Generate the complete dataset"""

print(f"Generating dataset with {self.num\_employees} employees over {self.days\_range} days...")

print(f"Malicious employees: {self.malicious\_employees} ({self.malicious\_ratio:.1%})")

data = []

start\_date = datetime.now() - timedelta(days=self.days\_range)

for emp\_id in list(self.employees.keys()):

is\_malicious = emp\_id in self.malicious\_employee\_ids

for day in range(self.days\_range):

current\_date = start\_date + timedelta(days=day)

*# Employee static info*

emp\_info = self.employees[emp\_id]

*# Update travel status*

travel\_activity = self.\_update\_trip\_status(emp\_id, current\_date.date())

is\_abroad = travel\_activity['is\_abroad'] == 1

*# Generate daily activities*

print\_activity = self.\_generate\_print\_activity(emp\_id, current\_date.date(), is\_malicious, is\_abroad)

burn\_activity = self.\_generate\_burn\_activity(emp\_id, current\_date.date(), is\_malicious, is\_abroad)

access\_activity = self.\_generate\_access\_activity(emp\_id, current\_date.date(), is\_malicious, is\_abroad)

*# Calculate risk travel indicator*

risk\_travel\_indicator = 0

if (travel\_activity['is\_abroad'] == 1 and

travel\_activity['is\_official\_trip'] == 0 and

travel\_activity['is\_hostile\_country\_trip'] == 1 and

(print\_activity['total\_printed\_pages'] > 0 or burn\_activity['total\_files\_burned'] > 0)):

risk\_travel\_indicator = 1

*# Combine all data*

row = {

'employee\_id': emp\_id,

'date': current\_date.date(),

'employee\_department': emp\_info['department'],

'employee\_campus': emp\_info['campus'],

'employee\_job': emp\_info['job'],

'employee\_seniority\_years': emp\_info['seniority\_years'],

'is\_contractor': emp\_info['is\_contractor'],

'employee\_classification': emp\_info['classification'],

'has\_foreign\_citizenship': emp\_info['foreign\_citizenship'],

'has\_criminal\_record': emp\_info['criminal\_record'],

'has\_medical\_history': emp\_info['medical\_history'],

'employee\_origin\_country': emp\_info['origin\_country'],

'is\_malicious': 1 if is\_malicious else 0, *# Target variable*

'risk\_travel\_indicator': risk\_travel\_indicator

}

*# Add all activity data*

row.update(print\_activity)

row.update(burn\_activity)

row.update(travel\_activity)

row.update(access\_activity)

data.append(row)

df = pd.DataFrame(data)

*# Convert trip\_day\_number to int where not null*

df['trip\_day\_number'] = df['trip\_day\_number'].astype('Int64') *# Nullable integer*

print(f"Dataset generated: {len(df)} records")

return df

*# Generate dataset and perform EDA*

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

generator = InsiderThreatDataGenerator(num\_employees=1000, days\_range=180)

df = generator.generate\_dataset()