

Student Information :
Promod Chandra Das-231002005
Chinmoy Debnath- 231902029
Course: AI Lab
Code:

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-
```

```
"""
```

AI Zero-Day Threat INDICATION & Anomaly Detection (FULL + FINAL)

Key upgrades:

- Auto dataset detection inside dataset/ (supports .csv OR .xlsx)
- If xlsx found, can export to dataset/data.csv automatically
- Clean UX (screen clear, compact outputs)
- Mode badge: REAL/DEMO
- Clean real-time SOC streaming (short WHY)
- Verdict line after run
- Attack-like label for explainability (not true attack classification)
- Presets + Advanced tuning
- Column preview & dataset sanity checks

Outputs:

- output/anomaly_results.csv
- output/top_alerts.csv
- output/anomaly_score_plot.png
- output/report.txt
- output/alerts.log

```
"""
```

```
import os  
import time  
from datetime import datetime  
from pathlib import Path
```

```
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt
```

```
from sklearn.ensemble import IsolationForest  
from sklearn.preprocessing import MinMaxScaler
```

```

# =====
# Utils
# =====
def clear_screen():
    os.system("cls" if os.name == "nt" else "clear")

def now_str():
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")


def safe_mkdir(path: str):
    os.makedirs(path, exist_ok=True)

def file_exists(path: str) -> bool:
    try:
        return bool(path) and os.path.isfile(path)
    except Exception:
        return False

def write_log(log_path: str, msg: str):
    with open(log_path, "a", encoding="utf-8") as f:
        f.write(f"[{now_str()}] {msg}\n")

def print_header(title: str):
    print("\n" + "=" * 78)
    print(title)
    print("=" * 78)

def press_enter():
    input("\nPress Enter to continue...")

def mode_badge(mode: str):
    if mode == "REAL_DATASET":
        print("\n" + "=" * 30)
        print(" MODE: REAL DATASET )
        print("=" * 30 + "\n")
    else:

```

```

print("\n" + "=" * 30)
print(" MODE: DEMO DATA ⚠ ")
print("=" * 30 + "\n")

# =====
# Demo Data
# =====
def generate_synthetic_telemetry(n=3000, seed=42):
    rng = np.random.default_rng(seed)

    cpu = np.clip(rng.normal(35, 10, n), 1, 95)
    mem = np.clip(rng.normal(45, 12, n), 1, 95)
    login_fail = np.clip(rng.poisson(1.2, n), 0, 15)
    outbound_kbps = np.clip(rng.lognormal(mean=5.2, sigma=0.35, size=n), 30, 5000)
    inbound_kbps = np.clip(rng.lognormal(mean=5.4, sigma=0.35, size=n), 30, 6000)
    unique_dst_ips = np.clip(rng.poisson(3.5, n), 0, 30)
    proc_spawn = np.clip(rng.poisson(4.0, n), 0, 40)
    port_scan_like = np.clip(rng.normal(0.05, 0.08, n), 0, 1)

    df = pd.DataFrame({
        "cpu_usage": cpu,
        "mem_usage": mem,
        "login_fail_count": login_fail,
        "outbound_kbps": outbound_kbps,
        "inbound_kbps": inbound_kbps,
        "unique_dst_ips": unique_dst_ips,
        "process_spawn_count": proc_spawn,
        "port_scan_like": port_scan_like,
    })

    # Inject anomalies (~2%)
    anomaly_idx = rng.choice(n, size=max(10, n // 50), replace=False)
    df.loc[anomaly_idx, "cpu_usage"] = np.clip(rng.normal(92, 4, len(anomaly_idx)), 70, 99)
    df.loc[anomaly_idx, "login_fail_count"] = rng.integers(20, 120, len(anomaly_idx))
    df.loc[anomaly_idx, "unique_dst_ips"] = rng.integers(40, 250, len(anomaly_idx))
    df.loc[anomaly_idx, "port_scan_like"] = np.clip(rng.normal(0.9, 0.08, len(anomaly_idx)), 0, 1)
    df.loc[anomaly_idx, "outbound_kbps"] = np.clip(
        rng.lognormal(mean=8.2, sigma=0.4, size=len(anomaly_idx)), 5000, 200000
    )

    df["ground_truth_demo"] = "Normal"
    df.loc[anomaly_idx, "ground_truth_demo"] = "Injected_Anomaly"
    return df

```

```

# =====
# Dataset discovery & loading
# =====
def find_dataset_file(dataset_dir: str, preferred_csv: str):
    """
    Returns: (chosen_path, kind)
    kind: "csv" | "xlsx" | None
    """
    if file_exists(preferred_csv):
        return preferred_csv, "csv"

    d = Path(dataset_dir)
    if not d.exists():
        return None, None

    # Prefer .csv (any) inside dataset/
    csvs = sorted([str(p) for p in d.glob("*.csv") if p.is_file()])
    if csvs:
        return csvs[0], "csv"

    # Then .xlsx
    xlsxs = sorted([str(p) for p in d.glob("*.xlsx") if p.is_file()])
    if xlsxs:
        return xlsxs[0], "xlsx"

    return None, None


def load_dataset_auto(dataset_dir: str, preferred_csv: str, log_path: str):
    """
    Auto-load:
    - If dataset/data.csv exists -> load csv
    - Else load first csv in dataset/
    - Else load first xlsx in dataset/ (read_excel)
    - Else demo data
    """
    chosen, kind = find_dataset_file(dataset_dir, preferred_csv)
    if not chosen:
        raw = generate_synthetic_telemetry(n=3000, seed=42)
        write_log(log_path, "Dataset not found. Using DEMO data.")
        return raw, "DEMO_DATA", None

```

```

try:
    if kind == "csv":
        raw = pd.read_csv(chosen)
        write_log(log_path, f"Loaded CSV: {chosen}")
        return raw, "REAL_DATASET", chosen
    else:
        # Excel
        raw = pd.read_excel(chosen)
        write_log(log_path, f"Loaded XLSX: {chosen}")
        return raw, "REAL_DATASET", chosen
except Exception as e:
    write_log(log_path, f"Failed to load dataset ({chosen}): {e}. Using DEMO.")
    raw = generate_synthetic_telemetry(n=3000, seed=42)
    return raw, "DEMO_DATA", None

```

```

def export_xlsx_to_csv_if_user_wants(xlsx_path: str, target_csv: str):
    """
    If user wants, export excel to csv for clean status 'FOUND dataset/data.csv'
    """
    clear_screen()
    print_header("XLSX DETECTED (Optional Export)")
    print("You have an Excel dataset file:")
    print(f" - {xlsx_path}")
    print("\nYour code default path expects:")
    print(f" - {target_csv}")
    print("\nIf you want, I can export XLSX -> CSV automatically.")
    ch = input("Export now? (y/n): ").strip().lower()
    if ch not in ["y", "yes"]:
        return False

    try:
        df = pd.read_excel(xlsx_path)
        safe_mkdir(str(Path(target_csv).parent))
        df.to_csv(target_csv, index=False)
        print(f"\n✅ Exported successfully: {target_csv}")
        press_enter()
        return True
    except Exception as e:
        print(f"\n❌ Export failed:", e)
        press_enter()
        return False

```

```

def preview_dataset(raw: pd.DataFrame):
    print_header("DATASET PREVIEW")
    print(f"Rows: {raw.shape[0]} | Columns: {raw.shape[1]}")
    cols = list(raw.columns)
    show_cols = cols[:12]
    print("Columns (first 12):", ", ".join([str(c) for c in show_cols]))
    if len(cols) > 12:
        print("... (more columns hidden)")
    print("\nSample (first 3 rows):")
    try:
        print(raw.head(3).to_string(index=False))
    except Exception:
        print(raw.head(3))
    press_enter()

# =====
# Preprocessing & Explain
# =====
LABEL_KEYWORDS = ["label", "class", "attack", "target", "ground_truth", "output", "y"]

def auto_detect_label_columns(df: pd.DataFrame):
    cols = []
    for c in df.columns:
        lc = str(c).lower()
        if any(kw in lc for kw in LABEL_KEYWORDS):
            cols.append(c)
    return cols

def make_numeric_features(raw: pd.DataFrame, drop_cols=None):
    df = raw.copy()

    if drop_cols:
        df = df.drop(columns=[c for c in drop_cols if c in df.columns], errors="ignore")

    non_numeric = df.select_dtypes(exclude=[np.number]).columns.tolist()
    if non_numeric:
        df = pd.get_dummies(df, columns=non_numeric, drop_first=True)

    df = df.select_dtypes(include=[np.number]).copy()
    df = df.replace([np.inf, -np.inf], np.nan).dropna()
    return df

```

```
def compute_baseline_stats(X_train_df: pd.DataFrame):
```

```
    mu = X_train_df.mean(axis=0)
```

```
    sigma = X_train_df.std(axis=0).replace(0, 1e-9)
```

```
    return mu, sigma
```

```
def top_deviation_features(feature_row: pd.Series, mu: pd.Series, sigma: pd.Series, top_k=3):
```

```
    z = ((feature_row - mu) / sigma).abs().sort_values(ascending=False)
```

```
    feats = z.head(top_k).index.tolist()
```

```
    return feats
```

```
# =====
```

```
# Risk + "Attack-like label" (Explain only)
```

```
# =====
```

```
def risk_level_from_score(score: float) -> str:
```

```
    if score < -0.25:
```

```
        return "CRITICAL"
```

```
    if score < -0.20:
```

```
        return "HIGH"
```

```
    if score < -0.10:
```

```
        return "MEDIUM"
```

```
    return "LOW"
```

```
def attack_like_label(risk: str) -> str:
```

```
    # NOTE: Not real attack classification; only analyst-friendly wording
```

```
    if risk in ["CRITICAL", "HIGH"]:
```

```
        return "Possible Intrusion"
```

```
    if risk == "MEDIUM":
```

```
        return "Suspicious Activity"
```

```
    return "Minor Anomaly"
```

```
# =====
```

```
# Core pipeline
```

```
# =====
```

```
def run_pipeline(raw: pd.DataFrame, settings: dict, paths: dict, dataset_mode: str, dataset_path: str):
```

```
    outdir = paths["outdir"]
```

```
    log_path = paths["log_path"]
```

```
    safe_mkdir(outdir)
```

```

label_cols = auto_detect_label_columns(raw)
features = make_numeric_features(raw, drop_cols=label_cols)

if features.shape[1] < 2:
    raise SystemExit("ERROR: Not enough numeric features after preprocessing.")

# Align raw with cleaned feature index
raw_aligned = raw.loc[features.index].copy()

n = len(features)
split = int(n * settings["train_ratio"])
split = max(10, min(split, n - 10))

scaler = MinMaxScaler()
X_all = scaler.fit_transform(features.values)

feat_names = features.columns.tolist()
X_all_df = pd.DataFrame(X_all, columns=feat_names, index=features.index)
X_train_df = X_all_df.iloc[:split]
X_test_df = X_all_df.iloc[split:]

mu, sigma = compute_baseline_stats(X_train_df)

model = IsolationForest(
    n_estimators=settings["n_estimators"],
    contamination=settings["contamination"],
    random_state=settings["seed"]
)
model.fit(X_train_df.values)

pred = model.predict(X_test_df.values)
score = model.decision_function(X_test_df.values)

result = raw_aligned.iloc[split:].copy()
result["anomaly_flag"] = np.where(pred == -1, "Anomaly", "Normal")
result["anomaly_score"] = score
result["risk_level"] = result["anomaly_score"].apply(risk_level_from_score)
result["threat_hint"] = result["risk_level"].apply(attack_like_label)

# Compact explain: top 3 deviating features
reasons = []
for i, idx in enumerate(X_test_df.index):
    if result.iloc[i]["anomaly_flag"] == "Anomaly":

```

```

        feats = top_deviation_features(X_test_df.loc[idx], mu, sigma,
top_k=settings["why_topk"])
        reasons.append(", ".join([f"{f} ↑" for f in feats]))
    else:
        reasons.append("")
    result["why_short"] = reasons

# Summary
normal_count = int((result["anomaly_flag"] == "Normal").sum())
anomaly_count = int((result["anomaly_flag"] == "Anomaly").sum())
risk_counts = result.loc[result["anomaly_flag"] == "Anomaly", "risk_level"].value_counts()
risk_map = {k: int(v) for k, v in risk_counts.items()}

# Save files
out_csv = os.path.join(outdir, "anomaly_results.csv")
result.to_csv(out_csv, index=False)

top_alerts = result[(result["anomaly_flag"] == "Anomaly") &
(result["risk_level"].isin(["MEDIUM", "HIGH", "CRITICAL"]))].copy()
top_alerts = top_alerts.sort_values("anomaly_score", ascending=True)
out_top = os.path.join(outdir, "top_alerts.csv")
top_alerts.to_csv(out_top, index=False)

plt.figure()
plt.plot(result["anomaly_score"].values)
plt.title("Anomaly Score (Detection Window) - Lower = More Suspicious")
plt.xlabel("Detection Sample Index")
plt.ylabel("IsolationForest Score")
out_png = os.path.join(outdir, "anomaly_score_plot.png")
plt.savefig(out_png, dpi=150, bbox_inches="tight")

# Verdict line
high = risk_map.get("HIGH", 0) + risk_map.get("CRITICAL", 0)
verdict = "✅ System looks normal"
if high > 0:
    verdict = f"⚠️ Potential threat detected ({high} HIGH/CRITICAL)"
elif risk_map.get("MEDIUM", 0) > 0:
    verdict = f"⚠️ Suspicious activity detected (MEDIUM anomalies)"
elif anomaly_count > 0:
    verdict = f"ℹ️ Minor anomalies detected"

# Report
report_path = os.path.join(outdir, "report.txt")
with open(report_path, "w", encoding="utf-8") as f:

```

```

f.write("AI Zero-Day Threat INDICATION & Anomaly Detection Report\n")
f.write(f"Generated: {now_str()}\n\n")
f.write("Goal-safe definition:\n")
f.write("- This INDICATES possible zero-day threats via UNSUPERVISED anomaly
detection.\n")
f.write("- It DOES NOT classify known attack types.\n\n")
f.write(f"DATA MODE: {dataset_mode}\n")
f.write(f"DATASET PATH: {dataset_path}\n\n")
f.write("MODEL: IsolationForest\n")
f.write(f"Params: n_estimators={settings['n_estimators']},
contamination={settings['contamination']}, train_ratio={settings['train_ratio']}\n")
f.write(f"Explain (WHY) topK={settings['why_topk']}\n\n")
f.write(f"SUMMARY: Normal={normal_count}, Anomaly={anomaly_count}\n")
f.write("Risk breakdown (anomalies):\n")
for k in ["CRITICAL", "HIGH", "MEDIUM", "LOW"]:
    f.write(f"{k}: {risk_map.get(k, 0)}\n")
f.write("\nVERDICT:\n")
f.write(verdict + "\n\n")
f.write("Output files:\n")
f.write(f"- {out_csv}\n- {out_top}\n- {out_png}\n- {report_path}\n- {log_path}\n")

```

Logging

```

write_log(log_path, f"MODE={dataset_mode} PATH={dataset_path}")
write_log(log_path, f"Rows={n} Train={split} Detect={n-split}")
write_log(log_path, f"Params: est={settings['n_estimators']}
contam={settings['contamination']} train_ratio={settings['train_ratio']}")
write_log(log_path, f"Summary: normal={normal_count} anomaly={anomaly_count}
risk={risk_map} verdict={verdict}")

```

Top show compact

```

top_show = result.sort_values("anomaly_score",
ascending=True).head(min(settings["display_top"], len(result)))

```

```

return {
    "result": result,
    "top_show": top_show,
    "normal": normal_count,
    "anomaly": anomaly_count,
    "risk_map": risk_map,
    "verdict": verdict,
    "out_csv": out_csv,
    "out_top": out_top,
    "out_png": out_png,
    "report_path": report_path,
}

```

```

# =====
# Real-time SOC streaming
# =====
def realtime_streaming(raw: pd.DataFrame, settings: dict, paths: dict, dataset_mode: str,
dataset_path: str):
    clear_screen()
    print_header("REAL-TIME STREAMING (SOC DEMO) - CLEAN VIEW")
    print("Streaming alerts one-by-one. (Short WHY only)\n")

    res = run_pipeline(raw, settings, paths, dataset_mode, dataset_path)
    df = res["result"]

    delay = settings["realtime_delay"]
    only_alerts = settings["realtime_only_alerts"]
    limit = settings["realtime_max_print"]

    printed = 0
    for idx, row in df.iterrows():
        if only_alerts and row["anomaly_flag"] != "Anomaly":
            continue

        if row["anomaly_flag"] == "Anomaly":
            print(f"[ALERT] IDX={idx} | RISK={row['risk_level']} | TYPE={row['threat_hint']} |
SCORE={row['anomaly_score']:.4f}")
            if row["why_short"]:
                print(f" WHY: {row['why_short']}")
            else:
                print(f"[OK]  IDX={idx} | SCORE={row['anomaly_score']:.4f}")

        printed += 1
        time.sleep(delay)
        if printed >= limit:
            print("\n[STOP] Max streaming lines reached (kept clean).")
            break

    print("\n[OK] Streaming done.")
    print("Check: output/top_alerts.csv for full list.")
    press_enter()

# =====
# Settings

```

```

# =====
def settings_menu(settings: dict):
    while True:
        clear_screen()
        print_header("SETTINGS (Presets + Advanced)")
        print(f"Current: contamination={settings['contamination']}, train_ratio={settings['train_ratio']},
est={settings['n_estimators']}, why_topk={settings['why_topk']}")
        print("1) Strict    (low false alarm)")
        print("2) Balanced  (recommended)")
        print("3) Aggressive (more alerts)")
        print("4) Advanced change")
        print("5) Back")
        ch = input("Choose: ").strip()

        if ch == "1":
            settings["contamination"] = 0.01
            settings["train_ratio"] = 0.75
            settings["n_estimators"] = 400
            settings["why_topk"] = 3
            print("[OK] Strict preset applied.")
            press_enter()

        elif ch == "2":
            settings["contamination"] = 0.02
            settings["train_ratio"] = 0.70
            settings["n_estimators"] = 400
            settings["why_topk"] = 3
            print("[OK] Balanced preset applied.")
            press_enter()

        elif ch == "3":
            settings["contamination"] = 0.05
            settings["train_ratio"] = 0.65
            settings["n_estimators"] = 500
            settings["why_topk"] = 3
            print("[OK] Aggressive preset applied.")
            press_enter()

        elif ch == "4":
            val = input("contamination (0.01-0.05) Enter=keep: ").strip()
            if val:
                try:
                    settings["contamination"] = float(val)
                except ValueError:

```

```

        pass

val = input("train_ratio (0.6-0.85) Enter=keep: ").strip()
if val:
    try:
        settings["train_ratio"] = float(val)
    except ValueError:
        pass

val = input("n_estimators (200-700) Enter=keep: ").strip()
if val:
    try:
        settings["n_estimators"] = int(val)
    except ValueError:
        pass

val = input("WHY topK (2-5) Enter=keep: ").strip()
if val:
    try:
        settings["why_topk"] = int(val)
    except ValueError:
        pass

val = input("Realtime delay seconds (0.0-0.3) Enter=keep: ").strip()
if val:
    try:
        settings["realtime_delay"] = float(val)
    except ValueError:
        pass

val = input("Realtime only alerts? (y/n) Enter=keep: ").strip().lower()
if val in ["y", "yes"]:
    settings["realtime_only_alerts"] = True
elif val in ["n", "no"]:
    settings["realtime_only_alerts"] = False

val = input("Realtime max lines (10-200) Enter=keep: ").strip()
if val:
    try:
        settings["realtime_max_print"] = int(val)
    except ValueError:
        pass

print("[OK] Updated.")

```

```

        press_enter()

    elif ch == "5":
        return

    else:
        print("Invalid choice.")
        time.sleep(0.4)

# =====
# Help / About
# =====
def show_help(dataset_dir: str, preferred_csv: str):
    clear_screen()
    print_header("HELP / ABOUT")
    print("Main Goal:")
    print("- INDICATE possible zero-day cyber threats via UNSUPERVISED anomaly detection.")
    print("- Learns NORMAL baseline -> flags novel deviations (signature-less).")
    print("\nImportant:")
    print("- This does NOT predict attack type names.")
    print("- 'TYPE' is just an analyst-friendly hint, not true classification.\n")

    print("Dataset rules:")
    print(f"- Default expected: {preferred_csv}")
    print(f"- Auto-detects any .csv or .xlsx inside: {dataset_dir}")
    print("\nOutput folder: output/")
    print("- anomaly_results.csv, top_alerts.csv, anomaly_score_plot.png, report.txt, alerts.log")
    press_enter()

# =====
# Dataset Wizard
# =====
def dataset_wizard(dataset_dir: str, preferred_csv: str, log_path: str):
    clear_screen()
    print_header("DATASET WIZARD (Fix DEMO Mode)")
    print("Fix DEMO Mode by putting a dataset in the dataset/ folder.\n")

    safe_mkdir(dataset_dir)

    print("Expected default file:")
    print(f"- {preferred_csv}")
    print("\nAuto-detect also supports:")

```

```

print("- any .csv in dataset/")
print("- any .xlsx in dataset/ (Excel)\n")

# show current files
files = sorted([p.name for p in Path(dataset_dir).glob("**") if p.is_file()])
if files:
    print("Current files in dataset/")
    for f in files:
        print("-", f)
else:
    print("dataset/ folder is empty.")

print("\nOptions:")
print("1) Keep default path (dataset/data.csv)")
print("2) Export XLSX -> data.csv (if you have .xlsx)")
print("3) Enter a custom path to load (full path)")
print("4) Back")
ch = input("\nChoose: ").strip()

if ch == "1":
    write_log(log_path, "Wizard: keep default path.")
    return preferred_csv

if ch == "2":
    # find first xlsx inside dataset/
    xlsxs = sorted([str(p) for p in Path(dataset_dir).glob("*.xlsx") if p.is_file()])
    if not xlsxs:
        print("\n❌ No .xlsx found in dataset/. Put an Excel file first.")
        press_enter()
        return preferred_csv
    ok = export_xlsx_to_csv_if_user_wants(xlsxs[0], preferred_csv)
    if ok:
        write_log(log_path, f"Wizard: exported {xlsxs[0]} -> {preferred_csv}")
        return preferred_csv

if ch == "3":
    p = input("Enter dataset path: ").strip().strip('"').strip("'")
    if not file_exists(p):
        print("\n❌ File not found. Keeping default.")
        press_enter()
        return preferred_csv
    write_log(log_path, f"Wizard: custom dataset path set: {p}")
    return p

```

```

return preferred_csv

# =====
# Main UI
# =====
def show_main_menu(dataset_dir: str, preferred_csv: str):
    chosen, kind = find_dataset_file(dataset_dir, preferred_csv)
    if chosen:
        status = f"FOUND  ({kind.upper()})"
    else:
        status = "NOT FOUND  (DEMO MODE)"

    print_header("AI Zero-Day Threat INDICATION & Anomaly Detection (FULL + FINAL)")
    print(f"Dataset folder: {dataset_dir}")
    print(f"Default CSV: {preferred_csv}")
    print(f"Status: {status}\n")

    print("1) Run Detection")
    print("2) Real-time Streaming (SOC demo)")
    print("3) Dataset Wizard (Fix DEMO mode)")
    print("4) Preview Dataset (columns + sample)")
    print("5) Settings")
    print("6) Help / About")
    print("7) Exit")

def main():
    dataset_dir = "dataset"
    preferred_csv = os.path.join(dataset_dir, "data.csv")
    outdir = "output"
    safe_mkdir(outdir)
    safe_mkdir(dataset_dir)

    log_path = os.path.join(outdir, "alerts.log")
    paths = {"outdir": outdir, "log_path": log_path}

    settings = {
        "contamination": 0.02,
        "train_ratio": 0.70,
        "seed": 42,
        "n_estimators": 400,

        "why_topk": 3,      # WHY short

```

```

"display_top": 10,    # compact display
"realtime_delay": 0.05,
"realtime_only_alerts": True,
"realtime_max_print": 40,
}

write_log(log_path, "=== Program started ===")

while True:
    clear_screen()
    show_main_menu(dataset_dir, preferred_csv)
    choice = input("\nEnter choice: ").strip()

    if choice == "1":
        clear_screen()
        raw, mode, used_path = load_dataset_auto(dataset_dir, preferred_csv, log_path)
        mode_badge("REAL_DATASET" if mode == "REAL_DATASET" else "DEMO_DATA")

        if used_path and used_path.lower().endswith(".xlsx") and (not
file_exists(preferred_csv)):
            print("NOTE: You are loading an XLSX dataset.")
            print("If you want status to show FOUND CSV, use Dataset Wizard -> Export XLSX ->
data.csv.\n")

        print_header("RUN DETECTION")
        print(f"[OK] MODE={mode} | shape={raw.shape}")

        res = run_pipeline(raw, settings, paths, "REAL_DATASET" if mode ==
"REAL_DATASET" else "DEMO_DATA", used_path or "DEMO")

        print("\n===== SUMMARY (Detection Window Only) =====")
        print(f"Normal: {res['normal']}")
        print(f"Anomaly: {res['anomaly']}")
        print("Risk breakdown (only anomalies):")
        for k in ["CRITICAL", "HIGH", "MEDIUM", "LOW"]:
            print(f" {k}: {res['risk_map'].get(k, 0)}")

        print("\nTop suspicious events (compact):")
        show_cols = ["anomaly_flag", "risk_level", "threat_hint", "anomaly_score"]
        print(res["top_show"][show_cols].to_string(index=True))

        print("\nVERDICT:", res["verdict"])

        print("\n[OK] Saved:")

```

```

print("-", res["out_csv"])
print("-", res["out_top"])
print("-", res["out_png"])
print("-", res["report_path"])
print("-", log_path)

press_enter()

elif choice == "2":
    raw, mode, used_path = load_dataset_auto(dataset_dir, preferred_csv, log_path)
    realtime_streaming(raw, settings, paths, "REAL_DATASET" if mode ==
"REAL_DATASET" else "DEMO_DATA", used_path or "DEMO")

elif choice == "3":
    preferred_csv = dataset_wizard(dataset_dir, preferred_csv, log_path)

elif choice == "4":
    raw, mode, used_path = load_dataset_auto(dataset_dir, preferred_csv, log_path)
    clear_screen()
    mode_badge("REAL_DATASET" if mode == "REAL_DATASET" else "DEMO_DATA")
    preview_dataset(raw)

elif choice == "5":
    settings_menu(settings)
    write_log(log_path, f"Settings updated: {settings}")

elif choice == "6":
    show_help(dataset_dir, preferred_csv)

elif choice == "7":
    write_log(log_path, "=== Program exit ===")
    clear_screen()
    print("Exit. Done.")
    break

else:
    print("Invalid choice.")
    time.sleep(0.4)

if __name__ == "__main__":
    main()

```