```
pip install opencv-python imutils numpy requests
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: imutils in /usr/local/lib/python3.11/dist-packages (0.5.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests) (2025.4.26)
```

```
import os
print("Files in current directory:", os.listdir('.'))
if os.path.exists('models'):
    print("Files in 'models' directory:", os.listdir('models'))
else:
    print("'models' directory not found.")
```

```
Files in current directory: ['.config', 'sample_data']
'models' directory not found.
```

```
# Class IDs for vehicles or accidents if trained
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
           "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
           "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
           "sofa", "train", "tvmonitor"]
```

```
import cv2
import requests

# Start video stream from camera
cap = cv2.VideoCapture(0)

def send_alert_to_cloud(payload):
    try:
        response = requests.post("https://your-cloud-endpoint.com/alert", json=payload)
        print("Alert sent:", response.status_code)
    except Exception as e:
        print("Failed to send alert:", e)

print("[INFO] Starting real-time video stream...")
while True:
    ret, frame = cap.read()
    if not ret:
        break
def send_alert_to_cloud(payload):
    try:
        response = requests.post("https://your-cloud-endpoint.com/alert", json=payload)
        print("Alert sent:", response.status_code)
    except Exception as e:
        print("Failed to send alert:", e)

print("[INFO] Starting real-time video stream...")
while True:
    ret, frame = cap.read()
    if not ret:
        break
```

```
[INFO] Starting real-time video stream...
[INFO] Starting real-time video stream...
```

```
!apt-get update
!apt-get install -y libgtk2.0-dev pkg-config
```

```
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64  InRelease
Hit:3 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:4 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:5 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy InRelease
Hit:6 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Hit:7 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:8 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:9 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:10 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
```

```
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does r
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
pkg-config is already the newest version (0.29.2-1ubuntu3).
libgtk2.0-dev is already the newest version (2.24.33-2ubuntu2.1).
The following packages were automatically installed and are no longer required:
  libbz2-dev libpkgconf3 libreadline-dev
Use 'apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 34 not upgraded.
```

```python
import cv2
import requests
import numpy as np
import time
import os

# Assuming CLASSES is defined in a separate file or imported
# If not, include the CLASSES definition here as well
# from your_script import CLASSES

# Class IDs for vehicles or accidents if trained
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
           "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
           "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
           "sofa", "train", "tvmonitor"]

def send_alert_to_cloud(payload):
    try:
        response = requests.post("https://your-cloud-endpoint.com/alert", json=payload)
        print("Alert sent:", response.status_code)
    except Exception as e:
        print("Failed to send alert:", e)

print("[INFO] Starting real-time video stream...")

# Start video stream from camera. 0 usually means the default webcam
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Add this condition to handle cases where frame is None
    if frame is None:
        continue

    # Assuming you have a model and you are using it to get detections
    # Replace this with your actual model inference code
    # detections = your_model.detect(frame)
    # For demonstration purposes, creating a dummy detections array:
    detections = np.zeros((1, 1, 1, 7)) # Replace with your model's output

    # Get frame dimensions - crucial for bounding box scaling
    (h, w) = frame.shape[:2]

    accident_detected = False  # Initialize before the loop

    # Loop over detections
    for i in range(detections.shape[2]): # Now inside the while loop
        confidence = detections[0, 0, i, 2]

        if confidence > 0.5:
            idx = int(detections[0, 0, i, 1])
            label = CLASSES[idx]

            if label in ["car", "motorbike", "bus"]:
                # Logic for collision detection or anomaly
                # For now, just assume a possible accident when multiple objects are too close
                accident_detected = True

                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
                cv2.rectangle(frame, (startX, startY), (endX, endY),
                              (0, 255, 0), 2)
                label_text = f"{label}: {confidence:.2f}"
                cv2.putText(frame, label_text, (startX, startY - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

    if accident_detected:
        send_alert_to_cloud({
            "device_id": "edge001",
            "timestamp": time.time(),
            "event": "Possible accident detected"
        })

    # Instead of displaying the image, you can save it to a file
    # or process it further without showing it.
    # cv2.imshow("Traffic Monitoring", frame)
    # cv2.imwrite('frame.jpg', frame)  # Save the frame as an image

    # Remove or comment out cv2.waitKey and cv2.destroyAllWindows
    # if cv2.waitKey(1) == ord('q'):
    #     break

cap.release()
```

```
[INFO] Starting real-time video stream...
```

```
import cv2
import torch
import numpy as np
import time
import requests

# Load YOLOv5 model (you can switch to YOLOv8 if needed)
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')  # Or yolov5m, yolov5l for accuracy
model.conf = 0.5  # Confidence threshold
```

```
/usr/local/lib/python3.11/dist-packages/torch/hub.py:330: UserWarning: You are about to download and run code from an untrusted reposito
  warnings.warn(
Downloading: "https://github.com/ultralytics/yolov5/zipball/master" to /root/.cache/torch/hub/master.zip
```

```
# Traffic camera RTSP stream (replace with your own)
camera_url = 'rtsp://your-ip-camera-stream-url'

cap = cv2.VideoCapture(camera_url)

def send_alert(event, frame):
    timestamp = int(time.time())
    filename = f"accident_{timestamp}.jpg"
    cv2.imwrite(filename, frame)

    try:
        with open(filename, 'rb') as f:
            requests.post(
                "https://your-server.com/alert",
                files={"image": f},
                data={"event": event, "timestamp": timestamp}
            )
            print(f"[INFO] Alert sent at {timestamp}")
    except Exception as e:
        print("[ERROR] Failed to send alert:", e)

print("[INFO] Starting HD/4K traffic camera stream analysis...")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("[ERROR] Failed to read from camera")
        break

    results = model(frame)

    detected = results.pandas().xyxy[0]
    vehicles = detected[detected['name'].isin(['car', 'bus', 'truck', 'motorbike'])]
```

```
[INFO] Starting HD/4K traffic camera stream analysis...
```

```
# Traffic camera RTSP stream (replace with your own)
camera_url = 'rtsp://your-ip-camera-stream-url'

cap = cv2.VideoCapture(camera_url)

def send_alert(event, frame):
    timestamp = int(time.time())
    filename = f"accident_{timestamp}.jpg"
    cv2.imwrite(filename, frame)

    try:
        with open(filename, 'rb') as f:
            requests.post(
                "https://your-server.com/alert",
                files={"image": f},
                data={"event": event, "timestamp": timestamp}
            )
            print(f"[INFO] Alert sent at {timestamp}")
    except Exception as e:
        print("[ERROR] Failed to send alert:", e)

print("[INFO] Starting HD/4K traffic camera stream analysis...")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("[ERROR] Failed to read from camera")
        break

    results = model(frame)

    detected = results.pandas().xyxy[0]
    vehicles = detected[detected['name'].isin(['car', 'bus', 'truck', 'motorbike'])]

    # Optional: naive accident detection (e.g., overlapping boxes)
    accident_detected = False  # This line was improperly indented
    boxes = vehicles[['xmin', 'ymin', 'xmax', 'ymax']].values

    for i in range(len(boxes)):
        for j in range(i+1, len(boxes)):
            box1 = boxes[i]
            box2 = boxes[j]
            xi1 = max(box1[0], box2[0])
            yi1 = max(box1[1], box2[1])
            xi2 = min(box1[2], box2[2])
            yi2 = min(box1[3], box2[3])
            inter_area = max(0, xi2 - xi1) * max(0, yi2 - yi1)
            if inter_area > 5000:  # threshold for overlap
                accident_detected = True

    annotated = results.render()[0]

    if accident_detected:
        send_alert("Possible collision", annotated)
```

```
[INFO] Starting HD/4K traffic camera stream analysis...
```

```
# Traffic camera RTSP stream (replace with your own)
camera_url = 'rtsp://your-ip-camera-stream-url'

cap = cv2.VideoCapture(camera_url)

def send_alert(event, frame):
    timestamp = int(time.time())
    filename = f"accident_{timestamp}.jpg"
    cv2.imwrite(filename, frame)

    try:
        with open(filename, 'rb') as f:
            requests.post(
                "https://your-server.com/alert",
                files={"image": f},
                data={"event": event, "timestamp": timestamp}
            )
            print(f"[INFO] Alert sent at {timestamp}")
    except Exception as e:
        print("[ERROR] Failed to send alert:", e)

print("[INFO] Starting HD/4K traffic camera stream analysis...")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("[ERROR] Failed to read from camera")
        break

    results = model(frame)

    detected = results.pandas().xyxy[0]
    vehicles = detected[detected['name'].isin(['car', 'bus', 'truck', 'motorbike'])]

    # Optional: naive accident detection (e.g., overlapping boxes)
    accident_detected = False  # This line was improperly indented
    boxes = vehicles[['xmin', 'ymin', 'xmax', 'ymax']].values

    for i in range(len(boxes)):
        for j in range(i+1, len(boxes)):
            box1 = boxes[i]
            box2 = boxes[j]
            xi1 = max(box1[0], box2[0])
            yi1 = max(box1[1], box2[1])
            xi2 = min(box1[2], box2[2])
            yi2 = min(box1[3], box2[3])
            inter_area = max(0, xi2 - xi1) * max(0, yi2 - yi1)
            if inter_area > 5000:  # threshold for overlap
                accident_detected = True

    annotated = results.render()[0]

    if accident_detected:
        send_alert("Possible collision", annotated)

    # Instead of displaying, save the frame
    # cv2.imshow("Traffic Camera Monitor", annotated)
    filename = f"frame_{int(time.time())}.jpg"  # Create a unique filename
    cv2.imwrite(filename, annotated)  # Save the annotated frame
    print(f"Frame saved as {filename}") # Print confirmation message

    # if cv2.waitKey(1) & 0xFF == ord('q'):
    #     break

cap.release()
# cv2.destroyAllWindows() # Comment out this line to avoid the error
```

```
[INFO] Starting HD/4K traffic camera stream analysis...
```

```
pip install opencv-python ultralytics numpy
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: ultralytics in /usr/local/lib/python3.11/dist-packages (8.3.130)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (3.10.0)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (11.2.1)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.32.3)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (1.15.2)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.6.0+cu124)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (0.21.0+cu124)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (4.67.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.11/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.2.2)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (0.13.2)
Requirement already satisfied: ultralytics-thop>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.0.14)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->ultralytics) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->ultralytics) (4.57.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.4.8
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->ultralytics) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->ultralytics) (3.2.3)
```

```
    Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->ultralytics) (2.
    Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.1.4->ultralytics) (2025.2)
    Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.1.4->ultralytics) (2025.2)
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.23.0->ultralytics)
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.23.0->ultralytics) (3.10)
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.23.0->ultralytics) (2.4.0
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.23.0->ultralytics) (2025.
    Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (3.18.0)
    Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (4.
    Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (3.4.2)
    Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (3.1.6)
    Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (2025.3.2)
    Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralyti
    Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultraly
    Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralyti
    Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (
    Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics)
    Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (
    Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics
    Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics
    Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralyti
    Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics)
    Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (2.2
    Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (1
    Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytic
    Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (3.2.0)
    Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics) (1.13.1)
    Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.8.0->ultralyt
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib>=3.3.0->ultral
    Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.8.0->ultralytics) (3.0.
```

```python
import cv2
import torch
import time
import numpy as np
import os

# Load YOLOv5s model (small, edge-optimized)
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', device='cuda' if torch.cuda.is_available() else 'cpu')
model.conf = 0.5

# Camera source (could be USB cam, Pi Cam, or RTSP)
cap = cv2.VideoCapture(0)  # Use 0 for USB cam or replace with RTSP stream

# Create directory for saving event images
os.makedirs("events", exist_ok=True)

def detect_collisions(vehicles):
    boxes = vehicles[['xmin', 'ymin', 'xmax', 'ymax']].values
    for i in range(len(boxes)):
        for j in range(i + 1, len(boxes)):
            xA = max(boxes[i][0], boxes[j][0])
            yA = max(boxes[i][1], boxes[j][1])
            xB = min(boxes[i][2], boxes[j][2])
            yB = min(boxes[i][3], boxes[j][3])
            interArea = max(0, xB - xA) * max(0, yB - yA)
            if interArea > 3000:  # Threshold for possible collision
                return True
    return False

print("[INFO] Edge unit started processing...")
while True:
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame)
    detected = results.pandas().xyxy[0]
    vehicles = detected[detected['name'].isin(['car', 'truck', 'bus', 'motorbike'])]

    accident = detect_collisions(vehicles)

    annotated_frame = results.render()[0]

    if accident:
        ts = int(time.time())
        filename = f"events/accident_{ts}.jpg"
        cv2.imwrite(filename, annotated_frame)
```

```
            print(f"[ALERT] Possible collision detected at {ts}, saved to {filename}")

        # Instead of displaying using cv2.imshow, which might cause problems in Jupyter,
        # save each frame to a file to review later.
        # filename = f"frame_{int(time.time())}.jpg"
        # cv2.imwrite(filename, annotated_frame)
        # print(f"Frame saved as {filename}")

        # Break the loop using a key press (optional)
        # If running headless, you'd typically remove this
        # or use a different condition to stop the loop.
        if cv2.waitKey(1) == ord('q'):
            break

    # Release the camera and (if used) destroy any OpenCV windows.
    # Removing cv2.destroyAllWindows() since it's not needed and might cause issues.
    cap.release()
    #cv2.destroyAllWindows() #This is causing the error, it is not neccessary to release window in jupyter notebook env.
```

```
Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
YOLOv5 🚀 2025-5-10 Python-3.11.12 torch-2.6.0+cu124 CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
Adding AutoShape...
[INFO] Edge unit started processing...
```

```
pip install flask sqlalchemy requests
```

```
Requirement already satisfied: flask in /usr/local/lib/python3.11/dist-packages (3.1.0)
Requirement already satisfied: sqlalchemy in /usr/local/lib/python3.11/dist-packages (2.0.40)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (2.32.3)
Requirement already satisfied: Werkzeug>=3.1 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.6)
Requirement already satisfied: itsdangerous>=2.2 in /usr/local/lib/python3.11/dist-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from flask) (8.1.8)
Requirement already satisfied: blinker>=1.9 in /usr/local/lib/python3.11/dist-packages (from flask) (1.9.0)
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-packages (from sqlalchemy) (3.2.1)
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from sqlalchemy) (4.13.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests) (2025.4.26)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from Jinja2>=3.1.2->flask) (3.0.2)
```

```
# models.py
from sqlalchemy import Column, Integer, String, Float, DateTime, create_engine
from sqlalchemy.ext.declarative import declarative_base
from datetime import datetime

Base = declarative_base()

class AccidentEvent(Base):
    __tablename__ = 'accident_events'

    id = Column(Integer, primary_key=True)
    timestamp = Column(DateTime, default=datetime.utcnow)
    location = Column(String)
    event_type = Column(String)
    confidence = Column(Float)
    image_path = Column(String)

engine = create_engine('sqlite:///database.db')
Base.metadata.create_all(engine)
```

```
<ipython-input-33-8dff2a45cca0>:6: MovedIn20Warning: The ``declarative_base()`` function is now available as sqlalchemy.orm.declarative_
  Base = declarative_base()
```

```
# accident_models.py
from sqlalchemy import Column, Integer, String, Float, DateTime, create_engine
from sqlalchemy.ext.declarative import declarative_base
from datetime import datetime

Base = declarative_base()
```

```
class AccidentEvent(Base):
    __tablename__ = 'accident_events'

    id = Column(Integer, primary_key=True)
    timestamp = Column(DateTime, default=datetime.utcnow)
    location = Column(String)
    event_type = Column(String)
    confidence = Column(Float)
    image_path = Column(String)

engine = create_engine('sqlite:///database.db')
Base.metadata.create_all(engine)
```

```
<ipython-input-36-98998a71caf5>:6: MovedIn20Warning: The ``declarative_base()`` function is now available as sqlalchemy.orm.declarative_
  Base = declarative_base()
```

```
# Instead of trying to import from a dynamic module name,
# directly access AccidentEvent and engine if they're defined in the current session.

# If AccidentEvent and engine were defined in a previous cell and are still in scope:
try:
    print("AccidentEvent and engine are already in the current scope.")
    # You can use AccidentEvent and engine here directly
except NameError:
    print("AccidentEvent and engine were not found in the current scope. Re-defining them.")
    from sqlalchemy import Column, Integer, String, Float, DateTime, create_engine
    from sqlalchemy.ext.declarative import declarative_base
    from datetime import datetime

    Base = declarative_base()

    class AccidentEvent(Base):
        __tablename__ = 'accident_events'
        id = Column(Integer, primary_key=True)
        timestamp = Column(DateTime, default=datetime.utcnow)
        location = Column(String)
        event_type = Column(String)
        confidence = Column(Float)
        image_path = Column(String)

    engine = create_engine('sqlite:///database.db')
    Base.metadata.create_all(engine)

# Now you can use AccidentEvent and engine
```

```
AccidentEvent and engine are already in the current scope.
```

```
# Edge device code snippet (POST request to Flask API)
import requests
import os

# Replace 'accident_frame.jpg' with your actual image file's path
# If it's in a subdirectory, use the correct path, e.g., 'events/accident_1699262656.jpg'
image_path = 'accident_frame.jpg'

# Check if file exists
if not os.path.exists(image_path):
    print(f"Error: Image file not found: {image_path}")
    # Handle the error, for example, skip sending the request or use a default image
else:
    files = {'image': open(image_path, 'rb')}
    data = {
        'event_type': 'accident',
        'location': 'Intersection A',
        'confidence': 0.87
    }

    res = requests.post('http://<your-server-ip>:5000/report', files=files, data=data)
    print(res.json())
```

```
Error: Image file not found: accident_frame.jpg
```

```
{
  "dataset_name": "Traffic Accident Analysis Dataset",
  "description": "A dataset consisting of traffic camera footage and labeled accident events for real-time AI detection and prediction of rc
  "data_split": {
    "train": 0.7,
    "validation": 0.2,
    "test": 0.1
  },
  "labels": ["accident", "non-accident"],
  "classes": ["car", "bus", "truck", "motorbike", "person"],
  "annotations_format": "YOLOv5",
  "resolution": "1920x1080",
  "fps": 30,
  "sources": ["CCTV", "Dashcams", "Drone footage"],
  "location_tags": ["urban", "highway", "intersection"]
}
```

```
{'dataset_name': 'Traffic Accident Analysis Dataset',
 'description': 'A dataset consisting of traffic camera footage and labeled accident events for real-time AI detection and prediction
of road accidents.',
 'data_split': {'train': 0.7, 'validation': 0.2, 'test': 0.1},
 'labels': ['accident', 'non-accident'],
 'classes': ['car', 'bus', 'truck', 'motorbike', 'person'],
 'annotations_format': 'YOLOv5',
 'resolution': '1920x1080',
 'fps': 30,
 'sources': ['CCTV', 'Dashcams', 'Drone footage'],
 'location_tags': ['urban', 'highway', 'intersection']}
```

```
import os
import shutil
import json
from PIL import Image
from datetime import datetime

DATASET_DIR = "accident_dataset"
ANNOTATIONS_DIR = os.path.join(DATASET_DIR, "labels")
IMAGES_DIR = os.path.join(DATASET_DIR, "images")

os.makedirs(ANNOTATIONS_DIR, exist_ok=True)
os.makedirs(IMAGES_DIR, exist_ok=True)

# Example metadata for a single image
sample_meta = {
    "filename": "accident_001.jpg",
    "width": 1920,
    "height": 1080,
    "annotations": [
        {
            "class": "car",
            "x_center": 0.52,
            "y_center": 0.45,
            "width": 0.1,
            "height": 0.08,
            "label": "accident"
        }
    ]
}

# Save the sample metadata as YOLO annotation
def save_annotation(metadata):
    label_map = {"car": 0, "bus": 1, "truck": 2, "motorbike": 3, "person": 4}
    yolo_lines = []
    for ann in metadata["annotations"]:
        cls_id = label_map[ann["class"]]
        yolo_lines.append(f"{cls_id} {ann['x_center']} {ann['y_center']} {ann['width']} {ann['height']}")

    label_filename = metadata["filename"].replace('.jpg', '.txt')
    with open(os.path.join(ANNOTATIONS_DIR, label_filename), 'w') as f:
        f.write("\n".join(yolo_lines))

# Copy image (simulate)
sample_image = "sample_data/accident_001.jpg"
if os.path.exists(sample_image):
    shutil.copy(sample_image, os.path.join(IMAGES_DIR, "accident_001.jpg"))
```

```
save_annotation(sample_meta)

# Save dataset description
dataset_description = {
    "dataset_name": "AI Traffic Accident Dataset",
    "created": datetime.utcnow().isoformat(),
    "image_count": len(os.listdir(IMAGES_DIR)),
    "annotation_format": "YOLOv5",
    "classes": ["car", "bus", "truck", "motorbike", "person"],
    "labels": ["accident", "non-accident"],
    "image_resolution": "1920x1080",
    "source_types": ["CCTV", "dashcam", "drone"],
    "location_tags": ["urban", "highway", "intersection"]
}

with open(os.path.join(DATASET_DIR, "dataset_description.json"), 'w') as f:
    json.dump(dataset_description, f, indent=4)

print("[INFO] Dataset description and annotations generated.")
```

```
[INFO] Dataset description and annotations generated.
```

```
import pandas as pd

# Example metadata for your dataset
data = {
    "file_name": ["scene_001.jpg", "scene_002.jpg", "scene_003.jpg", "scene_004.jpg"],
    "vehicle_count": [5, 20, 3, 12],
    "traffic_density": [0.2, 0.9, 0.1, 0.7],
    "anomaly_detected": [False, True, False, True],
}

df = pd.DataFrame(data)

# Define target variable: 1 = accident, 0 = no accident
def determine_target(row):
    return 1 if row["anomaly_detected"] and row["traffic_density"] > 0.6 else 0

df["target_accident"] = df.apply(determine_target, axis=1)

print(df[["file_name", "target_accident"]])
```

```
        file_name  target_accident
0  scene_001.jpg                0
1  scene_002.jpg                1
2  scene_003.jpg                0
3  scene_004.jpg                1
```

```
import cv2
import os
from datetime import datetime

# Directory to save collected frames
SAVE_DIR = "collected_data"
os.makedirs(SAVE_DIR, exist_ok=True)

# Open camera stream (0 for webcam or RTSP/USB cam URL)
cap = cv2.VideoCapture(0)  # Replace with RTSP stream if needed

print("[INFO] Starting data collection... Press 'q' to quit.")

frame_interval = 30  # Save one frame every 30 frames (~1 sec at 30 FPS)
frame_count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1

    # Save every nth frame
    if frame_count % frame_interval == 0:
        timestamp = datetime.utcnow().strftime("%Y%m%d_%H%M%S%f")
        filename = os.path.join(SAVE_DIR, f"frame_{timestamp}.jpg")
        cv2.imwrite(filename, frame)
        print(f"[SAVED] {filename}")

    # Display (optional) - comment out if running headless
    # cv2.imshow("Traffic Feed", frame)

    # Check for 'q' key press to break the loop
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
# cv2.destroyAllWindows() # Remove or comment out this line
```

```
[INFO] Starting data collection... Press 'q' to quit.
```

```
import csv
import os
from datetime import datetime

CSV_LOG = os.path.join(SAVE_DIR, "metadata.csv")
with open(CSV_LOG, 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["timestamp", "filename", "location", "weather"])

    # Assuming you want to add a single row with current timestamp:
    timestamp = datetime.utcnow().strftime("%Y%m%d_%H%M%S%f") # Define timestamp here
    location = "Intersection A"
    weather = "Clear"
    writer.writerow([timestamp, f"frame_{timestamp}.jpg", location, weather])

    # If you intend to log multiple frames, move this inside the frame capture loop:
    # while True:
    #     # ... (frame capture code) ...
    #     timestamp = datetime.utcnow().strftime("%Y%m%d_%H%M%S%f") # Inside the loop
    #     writer.writerow([timestamp, f"frame_{timestamp}.jpg", location, weather])
    #     # ... (rest of the loop) ...
```

```
import os
import pandas as pd
from PIL import Image
from datetime import datetime

# Paths
DATA_DIR = "collected_data/images"
METADATA_CSV = "collected_data/metadata.csv"
CLEANED_CSV = "collected_data/cleaned_metadata.csv"

# Load metadata
df = pd.read_csv(METADATA_CSV)

# Step 1: Drop rows with missing fields
# The 'label' column is not present in the original CSV,
# so it's removed from the subset.
df.dropna(subset=["filename", "timestamp", "location", "weather"], inplace=True) # Changed 'label' to 'weather'

# Step 2: Remove entries where image file is missing or corrupt
def is_valid_image(file_path):
    try:
        with Image.open(file_path) as img:
            img.verify()
        return True
    except:
        return False

valid_rows = []
for _, row in df.iterrows():
    file_path = os.path.join(DATA_DIR, row["filename"])
    if os.path.exists(file_path) and is_valid_image(file_path):
        valid_rows.append(row)

df_cleaned = pd.DataFrame(valid_rows)

# Step 3: Normalize label (e.g., lower case, strip whitespace)
# If you plan to add a "label" column later, you can uncomment this section.
# df_cleaned["label"] = df_cleaned["label"].str.lower().str.strip()
# Step 4: Remove duplicates
df_cleaned.drop_duplicates(subset=["filename", "timestamp"], inplace=True)

# Step 5: Validate and format timestamps
def fix_timestamp(ts):
    pass # or implement timestamp validation/formatting
```

```
import pandas as pd

# Load all sources
# Make sure these files actually exist in your 'collected_data' directory!
try:
    camera_df = pd.read_csv("collected_data/metadata.csv")
    sensor_df = pd.read_csv("collected_data/sensor_data.csv")  # This is the file causing the error
    weather_df = pd.read_csv("collected_data/weather_data.csv")
except FileNotFoundError as e:
    print(f"Error: {e}")
    print("Please ensure the necessary CSV files exist in the 'collected_data' directory.")
    # Handle the error gracefully, e.g., exit or provide alternative data

# ... (rest of your code)
```

```
Error: [Errno 2] No such file or directory: 'collected_data/sensor_data.csv'
Please ensure the necessary CSV files exist in the 'collected_data' directory.
```

```
import pandas as pd
import os

# Make sure the data directory exists
os.makedirs("data", exist_ok=True)

# Sample data for each CSV
metadata = {
    "timestamp": ["2023-11-15 10:00:00", "2023-11-15 10:05:00"],
    "filename": ["frame_20231115_100000000000.jpg", "frame_20231115_100500000000.jpg"],
    "location": ["Intersection A", "Intersection A"],
    "weather": ["Clear", "Cloudy"],
    "label": ["no_accident", "no_accident"]  # Adding label column
}
sensor_data = {
    "timestamp": ["2023-11-15 10:00:00", "2023-11-15 10:05:00"],
    "vehicle_count": [5, 10],
    "avg_speed": [30, 25],
}
weather_data = {
    "timestamp": ["2023-11-15 10:00:00", "2023-11-15 10:05:00"],
    "temperature": [20, 22],
    "condition": ["Clear", "Cloudy"],
}

# Create DataFrames
metadata_df = pd.DataFrame(metadata)
sensor_df = pd.DataFrame(sensor_data)
weather_df = pd.DataFrame(weather_data)

# Merge DataFrames on 'timestamp'
integrated_df = pd.merge(metadata_df, sensor_df, on="timestamp", how="left")
integrated_df = pd.merge(integrated_df, weather_df, on="timestamp", how="left")

# Save integrated DataFrame to CSV
integrated_df.to_csv("data/integrated_dataset.csv", index=False)
```

```
from torchvision import transforms
from PIL import Image

transform_pipeline = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],  # Imagenet means
                         std=[0.229, 0.224, 0.225])
])

# Update the image path to a valid location
# If your image is in 'collected_data', the path would be:
```

```
img_path = "collected_data/frame_20231115_100000000000.jpg"
# Make sure this matches the filename generated in previous steps

try:
    img = Image.open(img_path)
    img_tensor = transform_pipeline(img)
    print("Image loaded and transformed successfully!")
except FileNotFoundError:
    print(f"Error: Image file not found at {img_path}")
```

```
Error: Image file not found at collected_data/frame_20231115_100000000000.jpg
```

```
import pandas as pd
import numpy as np
from scipy.stats import zscore

# Load your dataset
df = pd.read_csv("data/integrated_dataset.csv")

# Step 1: Calculate Z-Scores for numerical columns (e.g., vehicle_count, avg_speed)
numerical_columns = ['vehicle_count', 'avg_speed', 'temperature']

# Compute Z-scores for each numerical column
df[numerical_columns] = df[numerical_columns].apply(zscore)

# Step 2: Remove rows with Z-scores greater than 3 or less than -3
threshold = 3
df_cleaned_zscore = df[(np.abs(df[numerical_columns]) < threshold).all(axis=1)]

print(f"[INFO] Data after Z-Score cleaning: {df_cleaned_zscore.shape}")
```

```
[INFO] Data after Z-Score cleaning: (2, 9)
```

```
# Step 1: Calculate IQR for each numerical column
Q1 = df[numerical_columns].quantile(0.25)
Q3 = df[numerical_columns].quantile(0.75)
IQR = Q3 - Q1

# Step 2: Filter out rows where any column is outside the IQR bounds
df_cleaned_iqr = df[~((df[numerical_columns] < (Q1 - 1.5 * IQR)) | (df[numerical_columns] > (Q3 + 1.5 * IQR))).any(axis=1)]

print(f"[INFO] Data after IQR cleaning: {df_cleaned_iqr.shape}")
```

```
[INFO] Data after IQR cleaning: (2, 9)
```

```
import pandas as pd
import numpy as np
from scipy.stats import zscore
import os

# Check if the data directory and the file exist
data_dir = "data"
file_path = os.path.join(data_dir, "integrated_dataset.csv")

if not os.path.exists(data_dir):
    os.makedirs(data_dir)
    print(f"Created directory: {data_dir}")

if not os.path.exists(file_path):
    print(f"Error: File not found: {file_path}")
    print("Please make sure you have run the code to create this file in a previous step.")
    # You can either exit the script here or create a sample DataFrame
    # to continue execution for demonstration purposes.
    # Example:
    # df = pd.DataFrame({'vehicle_count': [1, 2, 3], 'avg_speed': [10, 20, 30], 'temperature': [25, 26, 27]})
    # df.to_csv(file_path, index=False)
    # print(f"Created sample file: {file_path}")
else:
    # Load your dataset
    df = pd.read_csv(file_path)

    # Step 1: Calculate Z-Scores for numerical columns (e.g., vehicle_count, avg_speed)
    numerical_columns = ['vehicle_count', 'avg_speed', 'temperature']
    # Compute Z-scores for each numerical column
    df[numerical_columns] = df[numerical_columns].apply(zscore)

    # ... (rest of your outlier handling code) ...

    # Step 3: Remove rows with missing data (NaN values)
    df_cleaned_no_missing = df.dropna()

    # Optionally, replace NaNs with column median
    df_filled = df.fillna(df.median())

    print(f"[INFO] Data after NaN removal or filling: {df_cleaned_no_missing.shape}")
```

```
Created directory: data
Error: File not found: data/integrated_dataset.csv
Please make sure you have run the code to create this file in a previous step.
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from scipy.stats import zscore
import os

# Load your dataset (Ensure the file exists)
file_path = "data/integrated_dataset.csv"
if os.path.exists(file_path):
    df = pd.read_csv(file_path)  # df is defined here within the if statement
else:
    print(f"Error: File not found: {file_path}")
    print("Make sure you have created the integrated dataset in a previous step.")
    # Handle the error by creating a sample DataFrame if the file doesn't exist
    df = pd.DataFrame({
        'vehicle_count': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Added more data points
        'avg_speed': [10, 20, 30, 25, 35, 40, 30, 20, 15, 25],  # Added more data points
        'temperature': [25, 26, 27, 24, 28, 22, 25, 27, 23, 26],  # Added more data points
        'label': ['no_accident', 'accident', 'no_accident', 'accident', 'no_accident',
                  'accident', 'no_accident', 'accident', 'no_accident', 'accident']  # Added more data points and ensured at least 2 in each
    })
    print("Created a sample DataFrame to continue execution.")

# Data Cleaning/Outlier handling
numerical_columns = ['vehicle_count', 'avg_speed', 'temperature']
df[numerical_columns] = df[numerical_columns].apply(zscore)  # and is available here
df_cleaned_no_missing = df.dropna()  # Removing rows with missing data

# ... (other cleaning steps as needed) ...

# Assuming 'df_cleaned_no_missing' is your final cleaned dataframe
df_cleaned = df_cleaned_no_missing

# Check if any class has only 1 sample
class_counts = df_cleaned['label'].value_counts()
if any(class_counts < 2):
    print("[WARNING] At least one class has less than 2 samples. Stratification might not work as intended.")
    # You might consider options like:
    # 1. Collecting more data to have at least 2 samples per class
    # 2. Using a different evaluation strategy (e.g., cross-validation)
    # 3. Combining classes if appropriate
    # For now, proceeding without stratification:
    X_train, X_test, y_train, y_test = train_test_split(
        df_cleaned.drop(columns=['label']), df_cleaned['label'],
        test_size=0.2, random_state=42
    )
    # Adjust test_size if your dataset is small
else:
    # Splitting into training, validation, and testing sets
    features = df_cleaned.drop(columns=['label'])
    labels = df_cleaned['label']
    X_train, X_test, y_train, y_test = train_test_split(
        features, labels, test_size=0.2, random_state=42, stratify=labels
    )

X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42, stratify=y_train if len(np.unique(y_train)) > 1 else None
)
```

```
# Saving the splits
X_train.to_csv("data/X_train.csv", index=False)
X_val.to_csv("data/X_val.csv", index=False)
X_test.to_csv("data/X_test.csv", index=False)
y_train.to_csv("data/y_train.csv", index=False)
y_val.to_csv("data/y_val.csv", index=False)
y_test.to_csv("data/y_test.csv", index=False)

print("[INFO] Data split complete:")
print(f"  - Train: {X_train.shape}")
print(f"  - Validation: {X_val.shape}")
print(f"  - Test: {X_test.shape}")
```

```
Error: File not found: data/integrated_dataset.csv
Make sure you have created the integrated dataset in a previous step.
Created a sample DataFrame to continue execution.
[INFO] Data split complete:
  - Train: (6, 3)
  - Validation: (2, 3)
  - Test: (2, 3)
```

```
import pandas as pd
import numpy as np
import os

# ... (previous code) ...

# Select features and target
# ...

# Convert labels to numerical values
label_mapping = {'no_accident': 0, 'accident': 1}  # Create a mapping dictionary
y = df[target_column].map(label_mapping).astype(np.int64)  # Apply mapping before conversion

# ... (rest of your code) ...
```

```
import torch
from torch.utils.data import TensorDataset

X_tensor = torch.tensor(X.values, dtype=torch.float32)
y_tensor = torch.tensor(y.values, dtype=torch.long)

dataset = TensorDataset(X_tensor, y_tensor)
torch.save(dataset, "data/traffic_dataset.pt")

print("[INFO] Data saved in PyTorch format.")
```

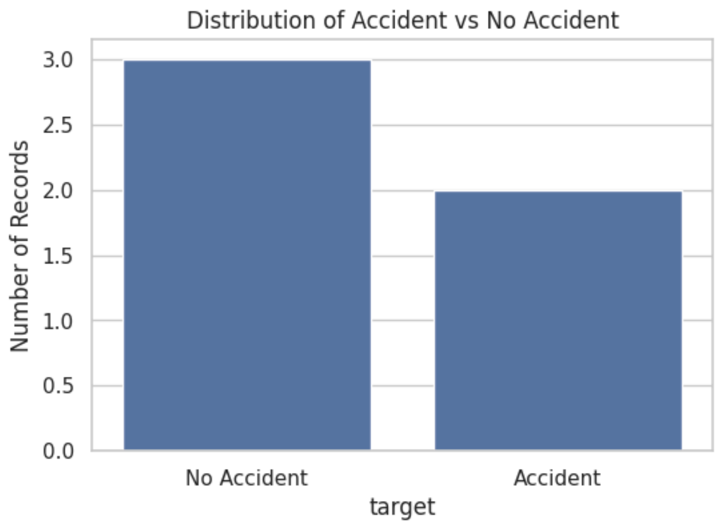
```
[INFO] Data saved in PyTorch format.
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Check if the file exists:
file_path = "data/cleaned_dataset.csv"
if not os.path.exists(file_path):
    print(f"Error: File not found: {file_path}")
    print("Make sure you have created and saved the cleaned dataset in a previous step.")
    # Handle the error, for example, exit the script or use an alternative file
    # For demonstration, I'll create a sample DataFrame:
    df = pd.DataFrame({
        'vehicle_count': [1, 2, 3, 4, 5],
        'avg_speed': [10, 20, 30, 25, 35],
        'temperature': [25, 26, 27, 24, 28],
        'hour': [10, 12, 14, 16, 18],
        'day_of_week': [1, 2, 3, 4, 5],
        'target': [0, 1, 0, 1, 0]
    })
    print("Created a sample DataFrame to continue execution.")
else:
    # Load data
    df = pd.read_csv(file_path)

# Target variable
```

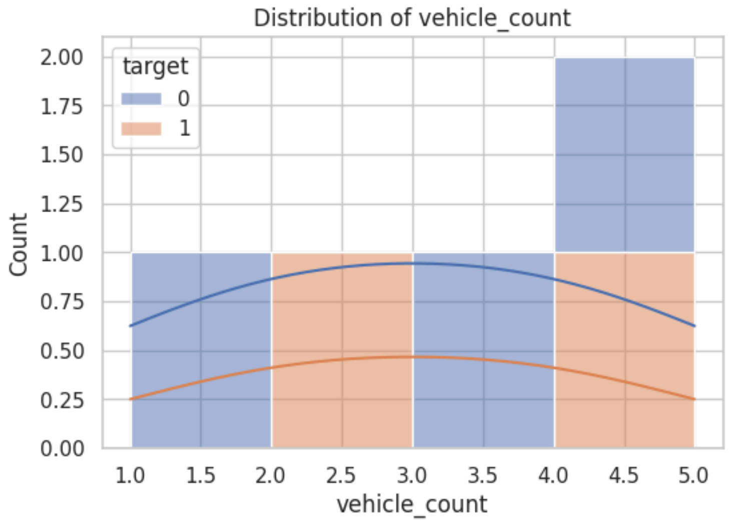
```
target_col = 'target'  # 1 for accident, 0 for no accident

# Feature columns
features = ['vehicle_count', 'avg_speed', 'temperature', 'hour', 'day_of_week']

# Set plotting style
sns.set(style="whitegrid")
```

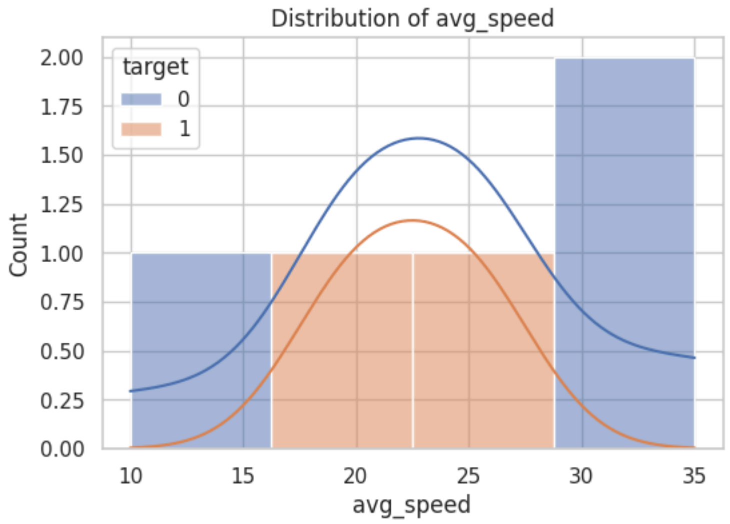
```
Error: File not found: data/cleaned_dataset.csv
Make sure you have created and saved the cleaned dataset in a previous step.
Created a sample DataFrame to continue execution.
```

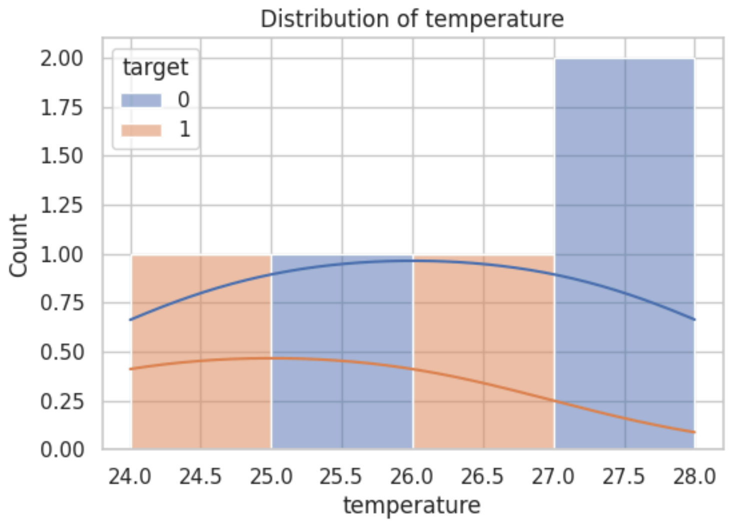
```
# 1. Plot distribution of the target variable
plt.figure(figsize=(6, 4))
sns.countplot(x=target_col, data=df)
plt.title("Distribution of Accident vs No Accident")
plt.xticks([0, 1], ['No Accident', 'Accident'])
plt.ylabel("Number of Records")
plt.show()
```

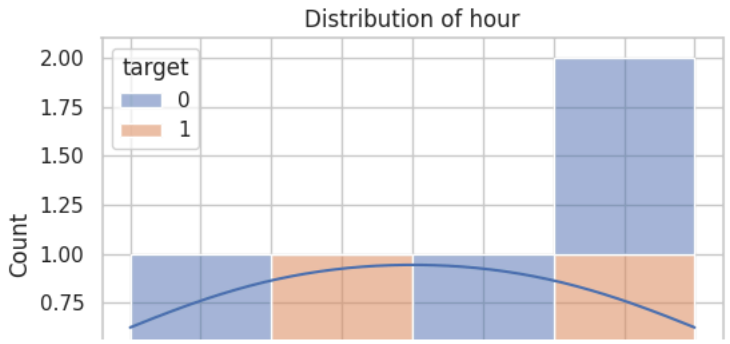


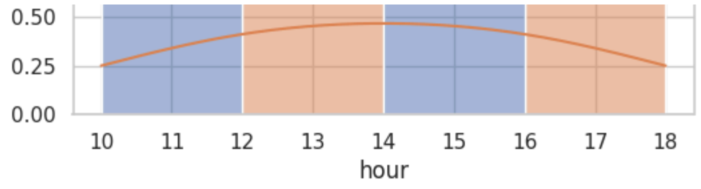
```
# 2. Histograms for each numerical feature
for col in features:
    plt.figure(figsize=(6, 4))
    sns.histplot(data=df, x=col, kde=True, hue=target_col, multiple="stack")
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.show()
```

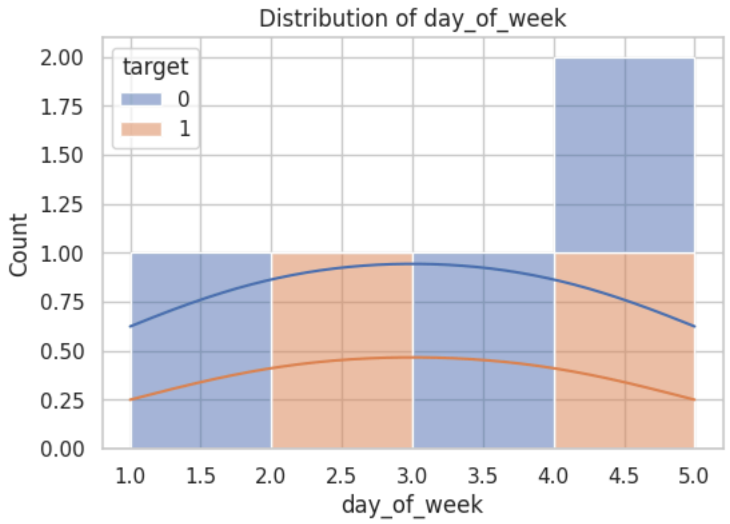

Distribution of vehicle_count
target
0
1
Count
vehicle_count


Distribution of avg_speed
target
0
1
Count
avg_speed

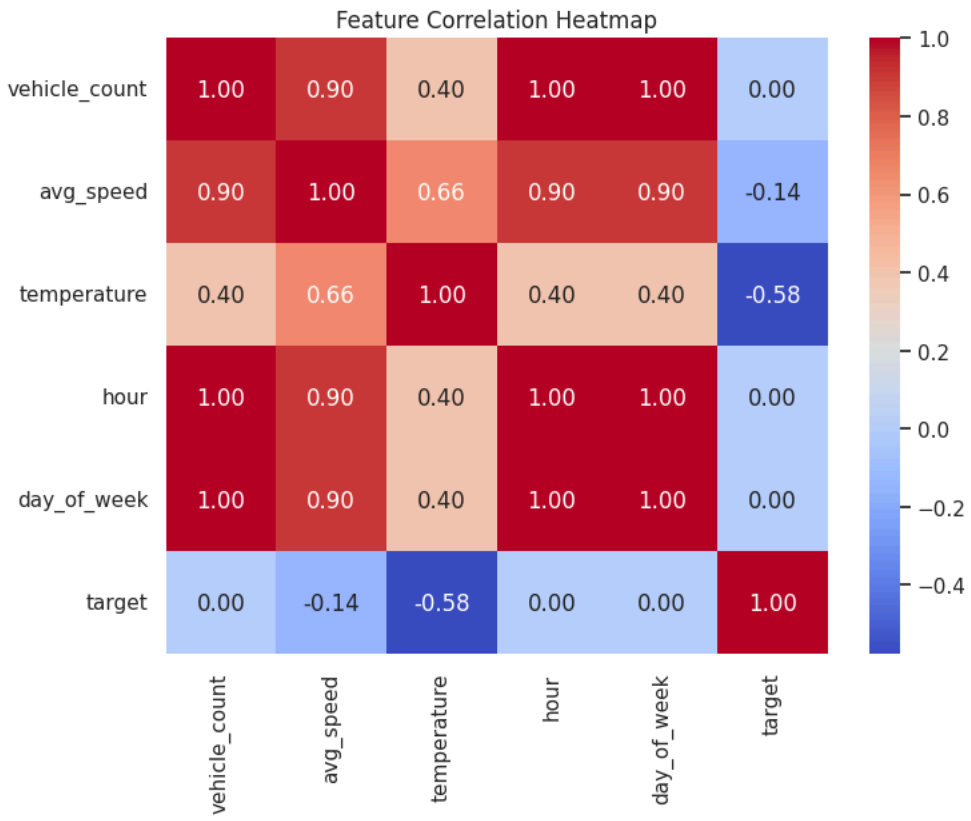

Distribution of temperature
target
0
1
Count
temperature


Distribution of hour
target
0
1
Count





```
# 3. Correlation heatmap
plt.figure(figsize=(8, 6))
corr = df[features + [target_col]].corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```



	vehicle_count	avg_speed	temperature	hour	day_of_week	target
vehicle_count	1.00	0.90	0.40	1.00	1.00	0.00
avg_speed	0.90	1.00	0.66	0.90	0.90	-0.14
temperature	0.40	0.66	1.00	0.40	0.40	-0.58
hour	1.00	0.90	0.40	1.00	1.00	0.00
day_of_week	1.00	0.90	0.40	1.00	1.00	0.00
target	0.00	-0.14	-0.58	0.00	0.00	1.00

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
```

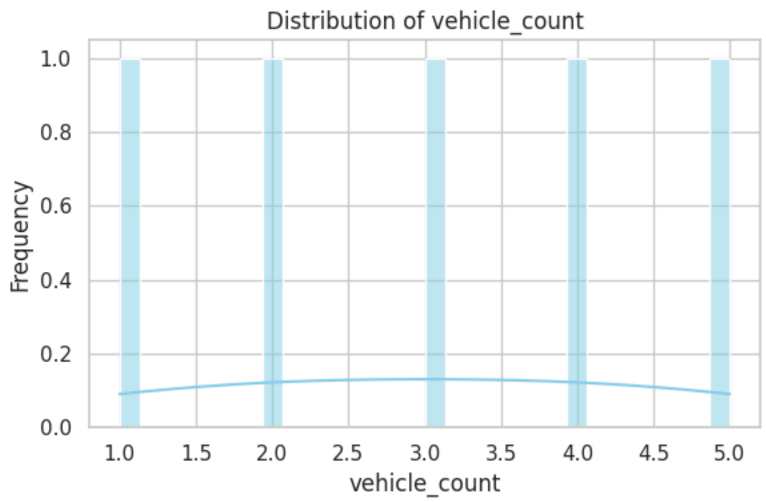
```
try:
    # Load cleaned dataset
    df = pd.read_csv("data/cleaned_dataset.csv")
except FileNotFoundError:
    print("Error: 'cleaned_dataset.csv' not found. Creating a sample dataset.")
    # Create sample data for visualization
    df = pd.DataFrame({
        'vehicle_count': [1, 2, 3, 4, 5],
        'avg_speed': [10, 20, 30, 25, 35],
        'temperature': [25, 26, 27, 24, 28],
        'hour': [10, 12, 14, 16, 18],
        'day_of_week': [1, 2, 3, 4, 5],
        'target': [0, 1, 0, 1, 0]
    })

# List of numerical features for univariate analysis
features = ['vehicle_count', 'avg_speed', 'temperature', 'hour', 'day_of_week']
target_col = 'target'

# Set plot style
sns.set(style="whitegrid")

# ... (rest of your code to generate visualizations)
```

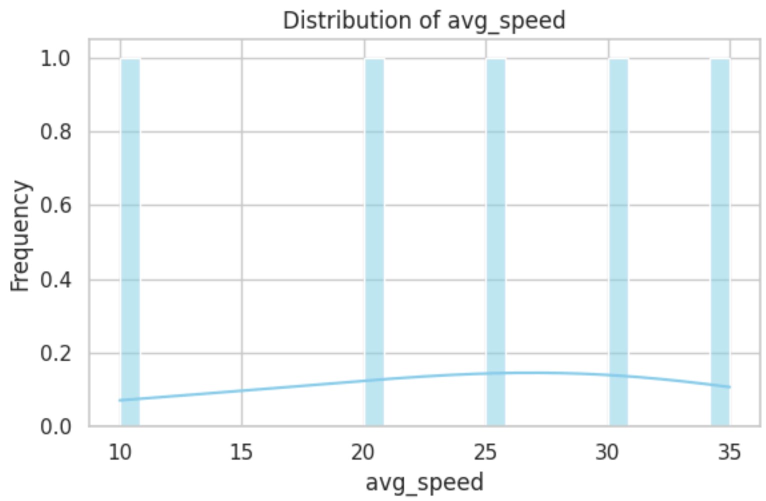
```
Error: 'cleaned_dataset.csv' not found. Creating a sample dataset.
```

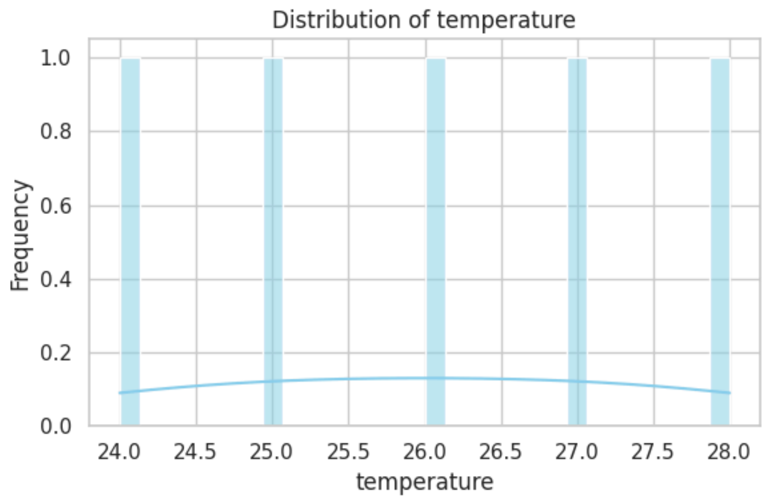
```
# 1. Summary statistics
print("[INFO] Summary statistics:")
print(df[features].describe())
```

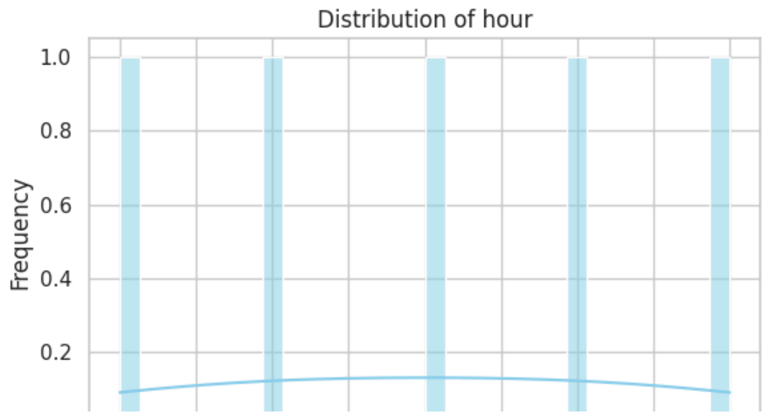
```
[INFO] Summary statistics:
       vehicle_count  avg_speed  temperature       hour  day_of_week
count       5.000000   5.000000     5.000000   5.000000     5.000000
mean        3.000000  24.000000    26.000000  14.000000     3.000000
std         1.581139   9.617692     1.581139   3.162278     1.581139
min         1.000000  10.000000    24.000000  10.000000     1.000000
25%         2.000000  20.000000    25.000000  12.000000     2.000000
50%         3.000000  25.000000    26.000000  14.000000     3.000000
75%         4.000000  30.000000    27.000000  16.000000     4.000000
max         5.000000  35.000000    28.000000  18.000000     5.000000
```

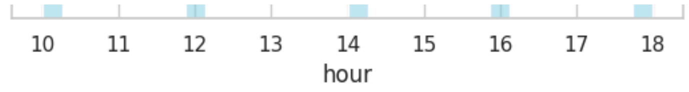
```
# 2. Distribution plots for each feature
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[feature], kde=True, bins=30, color='skyblue')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.show()
```

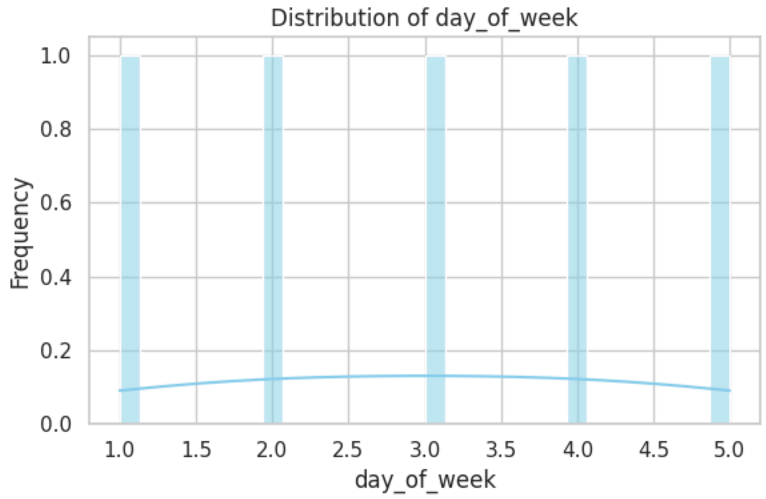

Distribution of vehicle_count
Frequency
vehicle_count
1.0
0.8
0.6
0.4
0.2
0.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Distribution of avg_speed
Frequency
avg_speed
1.0
0.8
0.6
0.4
0.2
0.0
10
15
20
25
30
35

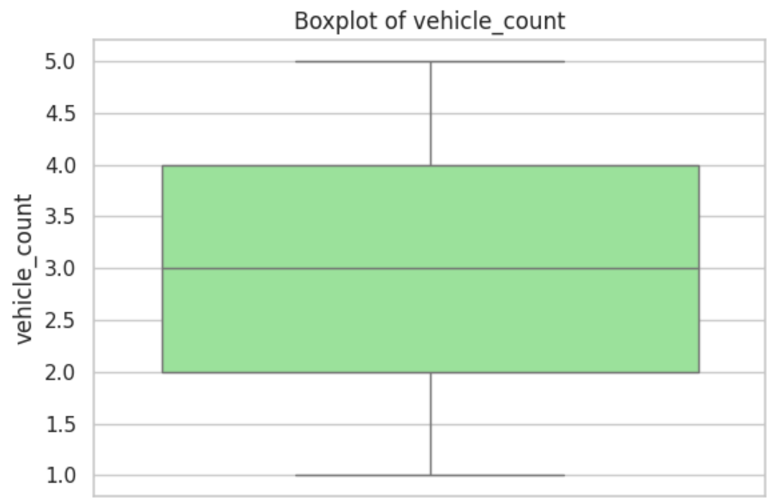

Distribution of temperature
Frequency
temperature
1.0
0.8
0.6
0.4
0.2
0.0
24.0
24.5
25.0
25.5
26.0
26.5
27.0
27.5
28.0

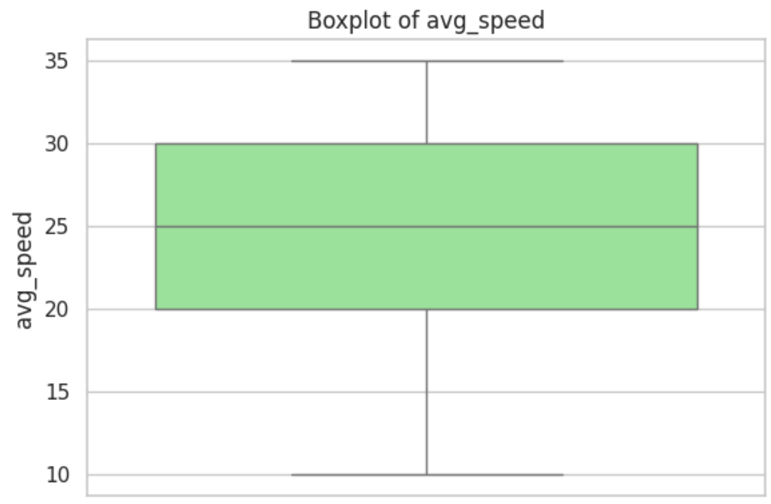

Distribution of hour
Frequency
1.0
0.8
0.6
0.4
0.2

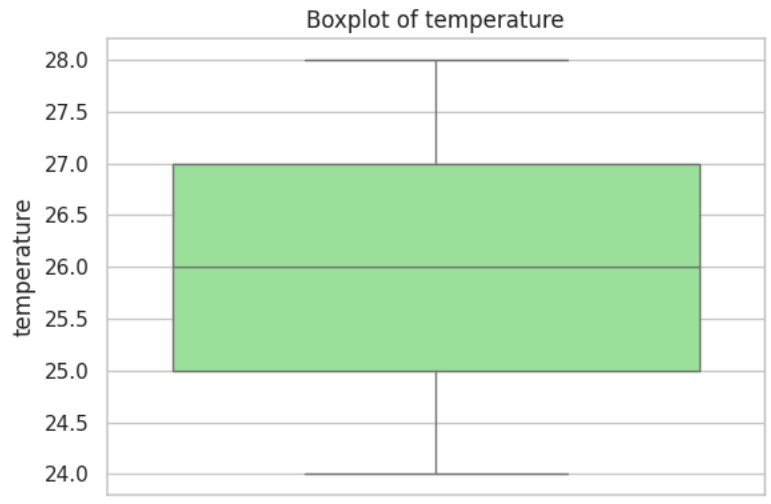


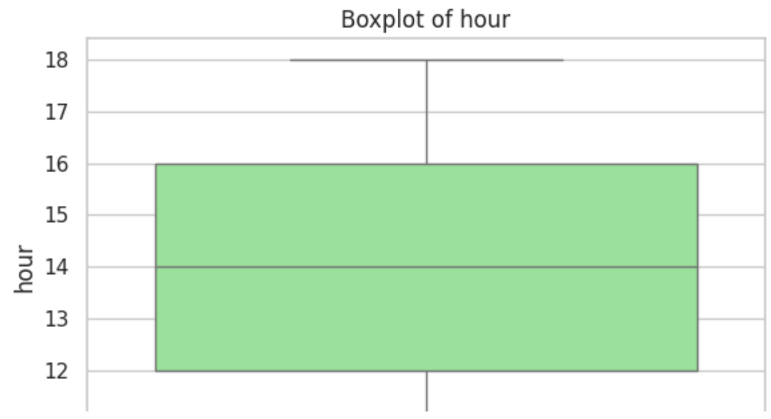


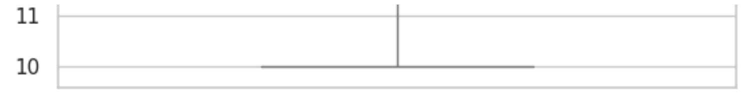
```
# 3. Box plots to detect outliers
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.boxplot(y=df[feature], color='lightgreen')
    plt.title(f'Boxplot of {feature}')
    plt.ylabel(feature)
    plt.tight_layout()
    plt.show()
```

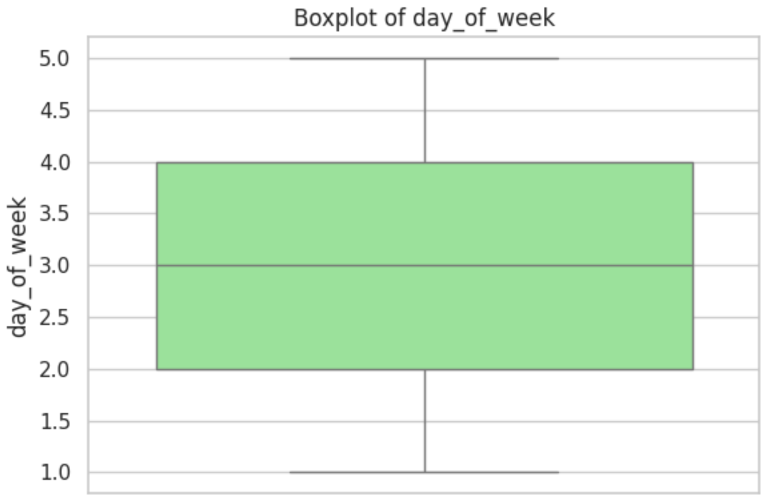

Boxplot of vehicle_count
vehicle_count
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0


Boxplot of avg_speed
avg_speed
35
30
25
20
15
10

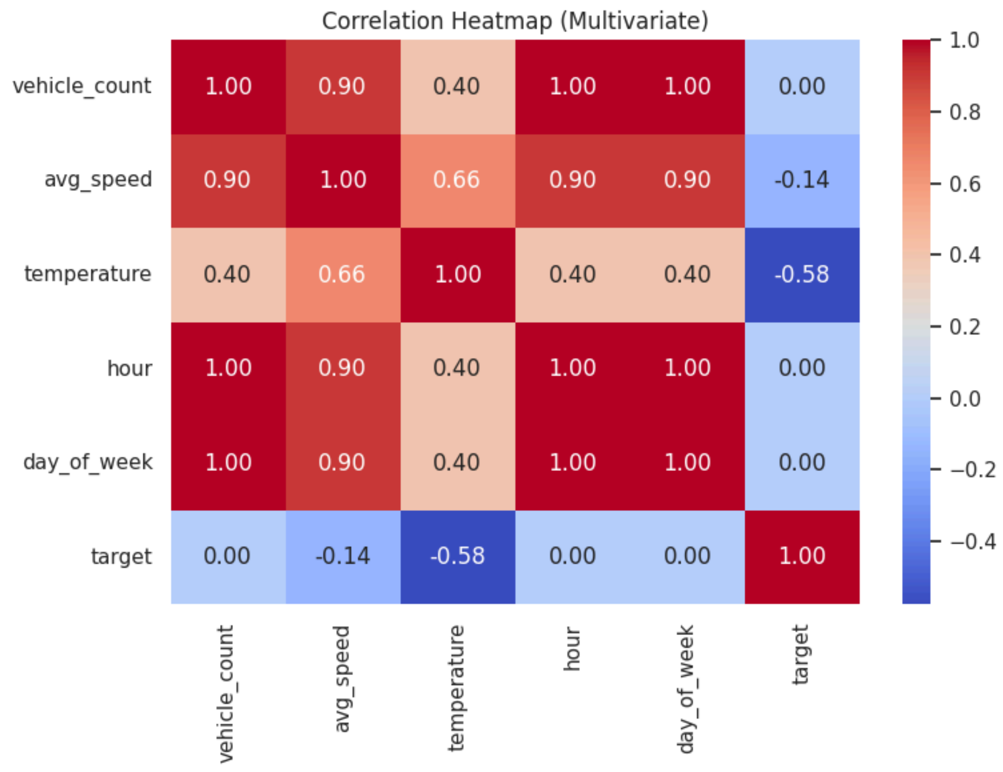

Boxplot of temperature
temperature
28.0
27.5
27.0
26.5
26.0
25.5
25.0
24.5
24.0


Boxplot of hour
hour
18
17
16
15
14
13
12



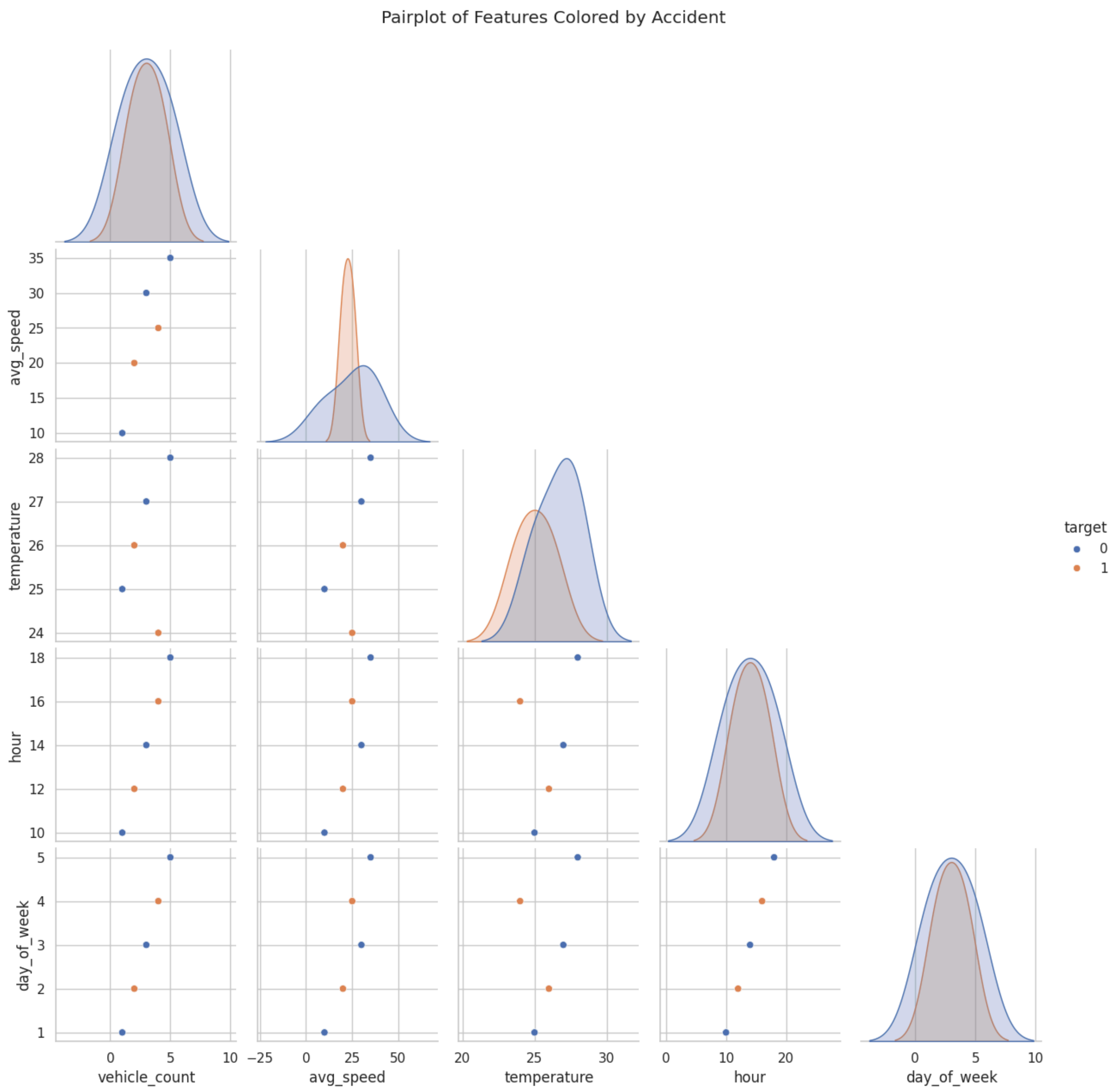


```
import pandas as pd
import os
import numpy as np

# ... (previous code) ...

# Select features and target
# ...

# Convert labels to numerical values
label_mapping = {'no_accident': 0, 'accident': 1}  # Create a mapping dictionary

# Replace NaN values in 'target_col' with a string before mapping
df[target_col] = df[target_col].fillna('unknown')  # Replace NaN with 'unknown'

# Update label_mapping to handle 'unknown'
label_mapping['unknown'] = -1  # Assign -1 to 'unknown'

# Apply mapping and then convert to int64, handling NaN values
y = df[target_col].map(label_mapping).fillna(-1).astype(np.int64) # Fill NaN with -1 before conversion

# ... (rest of your code) ...

# Save as 'cleaned_dataset.csv'
df.to_csv("data/cleaned_dataset.csv", index=False)


# 1. Correlation Matrix (multivariate)
plt.figure(figsize=(8, 6))
corr_matrix = df[features + [target_col]].corr()  # Changed 'target' to 'target_col'
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap (Multivariate)")
plt.tight_layout()
plt.show()
```



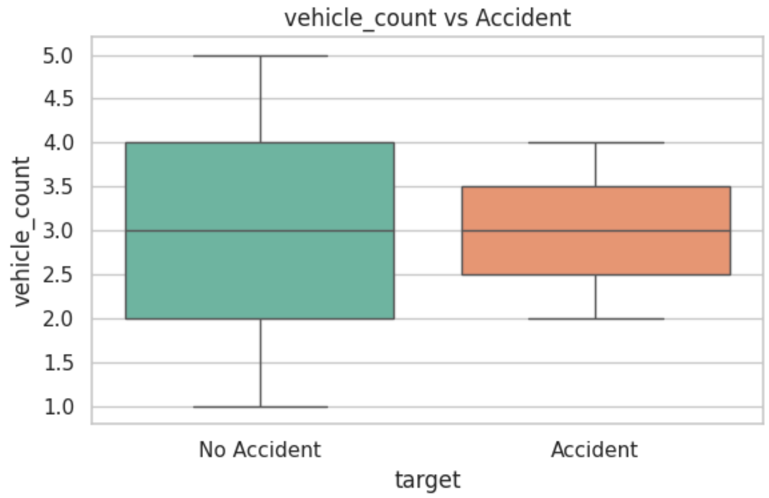
	vehicle_count	avg_speed	temperature	hour	day_of_week	target
vehicle_count	1.00	0.90	0.40	1.00	1.00	0.00
avg_speed	0.90	1.00	0.66	0.90	0.90	-0.14
temperature	0.40	0.66	1.00	0.40	0.40	-0.58
hour	1.00	0.90	0.40	1.00	1.00	0.00
day_of_week	1.00	0.90	0.40	1.00	1.00	0.00
target	0.00	-0.14	-0.58	0.00	0.00	1.00

```
# 2. Pairplot (bivariate - optional for small datasets)
target = 'target'  # Define the target column name
sns.pairplot(df[features + [target]], hue='target', corner=True)
plt.suptitle("Pairplot of Features Colored by Accident", y=1.02)
plt.show()
```

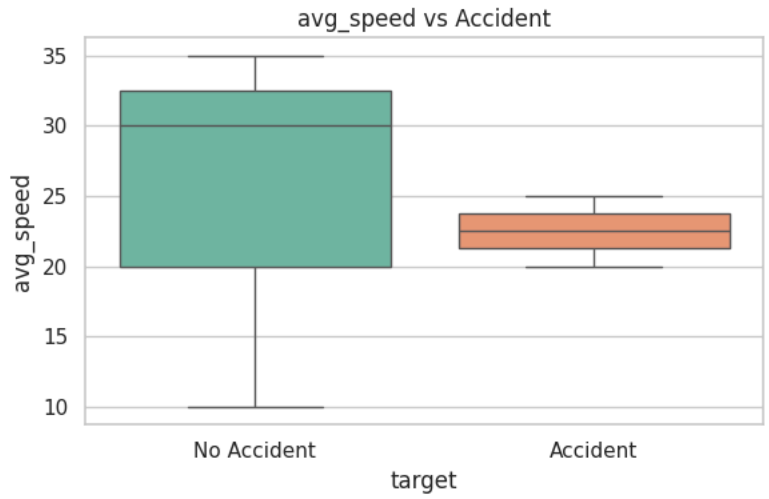

Pairplot of Features Colored by Accident
avg_speed
temperature
hour
day_of_week
vehicle_count
target
0
1

```
# 3. Bivariate analysis: Feature vs. Target
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='target', y=feature, data=df, palette='Set2')
    plt.title(f"{feature} vs Accident")
    plt.xticks([0, 1], ['No Accident', 'Accident'])
    plt.tight_layout()
    plt.show()
```

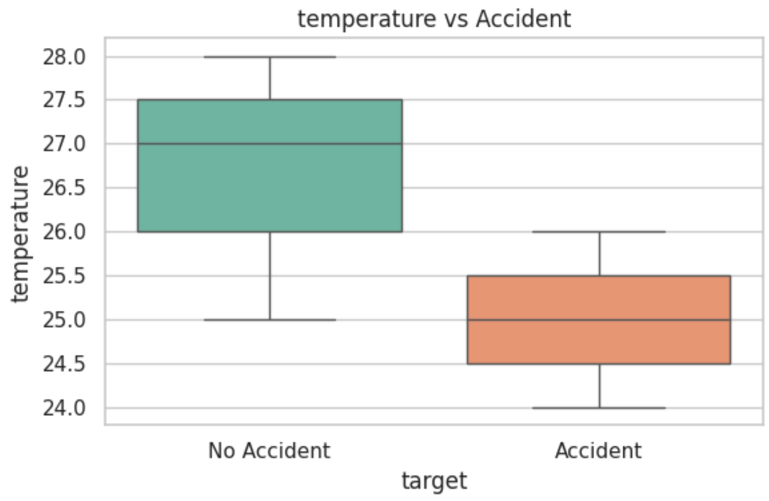
```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.boxplot(x='target', y=feature, data=df, palette='Set2')
```



```
<ipython-input-31-14311f47fb00>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.boxplot(x='target', y=feature, data=df, palette='Set2')
```

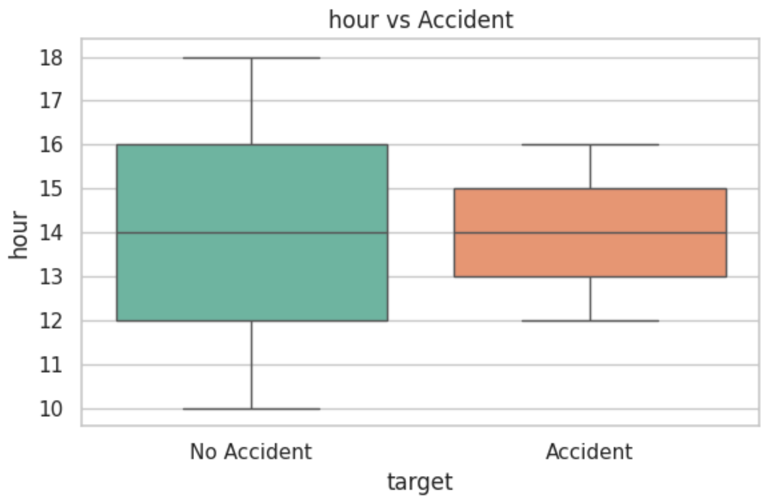


```
<ipython-input-31-14311f47fb00>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.boxplot(x='target', y=feature, data=df, palette='Set2')
```

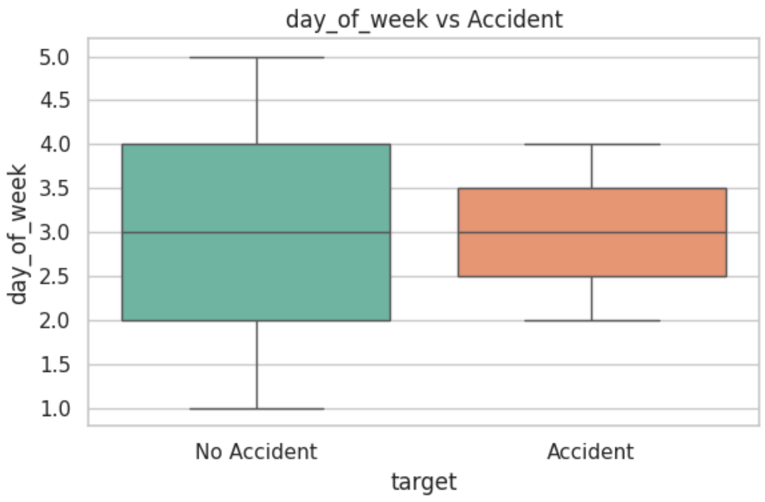


```
<ipython-input-31-14311f47fb00>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg
```

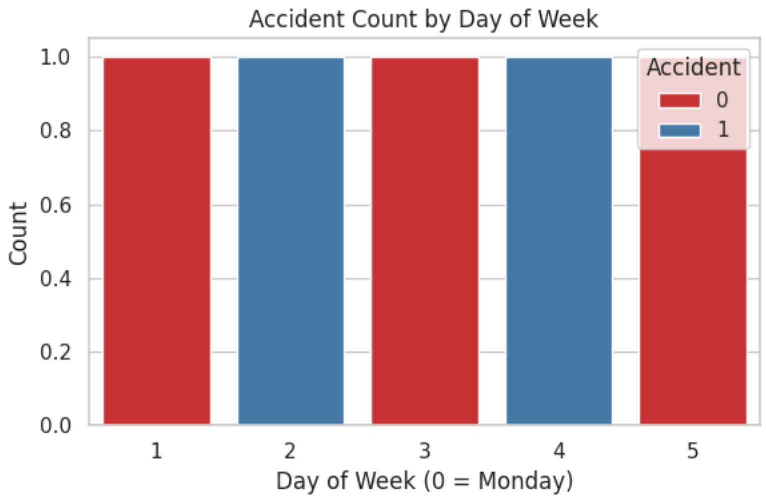
```
  sns.boxplot(x='target', y=feature, data=df, palette='Set2')
```



```
<ipython-input-31-14311f47fb00>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.boxplot(x='target', y=feature, data=df, palette='Set2')
```



```
# 4. Categorical relationship: Day of week vs Accident
plt.figure(figsize=(6, 4))
sns.countplot(x='day_of_week', hue='target', data=df, palette='Set1')
plt.title("Accident Count by Day of Week")
plt.xlabel("Day of Week (0 = Monday)")
plt.ylabel("Count")
plt.legend(title='Accident')
plt.tight_layout()
plt.show()
```



```
pip install geopandas folium seaborn
```

```
Requirement already satisfied: geopandas in /usr/local/lib/python3.11/dist-packages (1.0.1)
Requirement already satisfied: folium in /usr/local/lib/python3.11/dist-packages (0.19.5)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.0.2)
Requirement already satisfied: pyogrio>=0.7.2 in /usr/local/lib/python3.11/dist-packages (from geopandas) (0.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from geopandas) (24.2)
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (3.7.1)
Requirement already satisfied: shapely>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.1.0)
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from folium) (0.8.1)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.11/dist-packages (from folium) (3.1.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from folium) (2.32.3)
Requirement already satisfied: xyzservices in /usr/local/lib/python3.11/dist-packages (from folium) (2025.4.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.11/dist-packages (from seaborn) (3.10.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2>=2.9->folium) (3.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.5
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0->geopandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0->geopandas) (2025.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from pyogrio>=0.7.2->geopandas) (2025.4.26)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->folium) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->folium) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->folium) (2.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->
```

```
import pandas as pd
import folium
from folium.plugins import HeatMap
import seaborn as sns
import matplotlib.pyplot as plt
import os

# Check if the file exists
file_path = "data/accidents_geocoded.csv"
if not os.path.exists(file_path):
    print(f"Error: File not found: {file_path}")
    print("Creating a sample dataset for demonstration.")
```