Gartner Occupancy Project
System Maintenance Document

Copyright and Warranty Information

Table of Contents

Integration with Cisco overview

Project Overview

The Occupancy and Utilization Insights project set out to help Gartner make the most of its office space in Sydney. Our goal was to create a tool that could give real-time insights into how different areas are being used. Working with Cisco and Korus Australia, we combined Cisco's tech with custom computer vision tools to track and analyze workspace usage, providing actionable information that can guide decisions on office layout, resource allocation, and improving the overall work environment.

1. How We Got There: Project Phases

To keep things organized, we broke the project down into six key phases:

1. Getting the Idea Right

In this first phase, we met with Gartner to get a clear picture of the project goals, ensuring everyone was aligned on what success would look like. This included defining key outcomes, like real-time occupancy tracking and people detection, to meet Gartner's needs for a responsive, data-driven tool.

2. Setting Up the Tools and Plan

We selected the technology stack, choosing Cisco for connectivity and the YOLOv8 model for people detection, known for its precision. We planned out how data would flow from cameras to cloud storage, making sure it was secure, accessible, and real-time.

3. Developing People Detection

Using YOLOv8, we trained a model to detect people across different office settings. We fine-tuned it to work well in various conditions, like changes in lighting or office layouts. This setup allowed us to capture accurate, real-time data on individual desk and room usage.

4. Building Group Detection

We went a step further by developing a model to detect groups of people based on their proximity and positions. This helped us understand how people naturally gather and interact, providing insights that can guide office layout changes to encourage collaboration.

5. Bringing It All Together with a Dashboard

Our team created a Power BI dashboard to present all the data in a user-friendly way. Now, Gartner's team can easily see metrics like usage trends, peak times, and group dynamics, all at a glance.

6. Looking to the Future

We wrapped up by exploring how this system could grow. There's potential to add predictive features, like forecasting office usage trends, and even integrating environmental data, like lighting and temperature, for a fuller view of office comfort.

1.2. What We Achieved

With the project complete, here's what we were able to deliver:

- **Real-Time Monitoring of Office Space:** Using cameras and Cisco's technology, we set up a system that tracks room and desk usage in real time. Now, Gartner can see exactly when and how different spaces are being used.
- **Detailed Group and Zone Insights:** By detecting and analyzing groups, we provided a deeper understanding of how employees interact within various office zones. This helps in designing spaces that promote effective collaboration.
- **Visual Analytics on the Dashboard:** The PowerBI dashboard offers a clear picture of office utilization, showing:
    1. **Utilization KPIs:** Key metrics that highlight which spaces are used the most and when.
    2. **Seat-Level Details:** Specific data on individual workspaces.
    3. **Time-Based Trends:** Insights into daily, weekly, and monthly patterns.
- **Better Decision Support:** With these insights, Gartner can now make data-driven decisions about how to arrange office spaces to better suit employee needs and improve productivity.

Looking Ahead: Future Possibilities

This pilot project was just beginning. There's a lot of potential to expand this tool further. Here are some ideas:

- Predictive Analytics: We could add predictive features to anticipate office usage patterns, helping Gartner plan even more efficiently.
- Extended Monitoring: Incorporating environmental data like lighting and temperature could provide a fuller picture of office comfort and efficiency.
- System Integration: Integrating with facility management systems would create a seamless process for managing office spaces from end to end.

2. Technical Highlights: Data Collection Evolution

Our approach to collecting images evolved over the course of the project as we refined our methods to make the data collection process more efficient and responsive.
Initially, we relied on a Python script that took snippets from pre-recorded videos of the office space. This method provided a steady stream of images, allowing us to conduct early testing and analysis. However, it had limitations. The process was time-consuming, and the images weren't always up-to-date with the current office conditions, which could limit the model's effectiveness in real-time scenarios.
To address this, we transitioned to a more direct method. By connecting to the camera's IP address, we could capture live images directly from the source. This new setup allowed us to take screenshots at specified intervals, ensuring that we always had the latest images for our analysis. Each captured image was automatically uploaded to AWS S3 buckets, where they were securely stored and easily accessible for further processing.

```python
import cv2
import time
import boto3
from datetime import datetime
import os

# AWS S3 Configuration
s3_bucket_name = 'your-s3-bucket-name'
s3_folder_name = 'cctv_snapshots'
aws_region = 'your-aws-region'

# Initialize S3 client
s3 = boto3.client('s3', region_name=aws_region)

camera_sources = [
    "rtsp://username:password@camera1_ip_address",
    "rtsp://username:password@camera2_ip_address",
]

def capture_and_upload_snapshot(camera_url, camera_id):
    cap = cv2.VideoCapture(camera_url)

    if not cap.isOpened():
        print(f"Unable to connect to camera {camera_id}")
        return

    ret, frame = cap.read()

    if ret:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"camera_{camera_id}_{timestamp}.jpg"
        local_path = os.path.join("/tmp", filename)
        cv2.imwrite(local_path, frame)

        s3_path = f"{s3_folder_name}/{filename}"
        try:
            s3.upload_file(local_path, s3_bucket_name, s3_path)
            print(f"Snapshot from camera {camera_id} uploaded to {s3_bucket_name}/{s3_path}")
        except Exception as e:
            print(f"Failed to upload snapshot from camera {camera_id} to S3: {e}")
        finally:
            os.remove(local_path)
    else:
        print(f"Failed to capture snapshot from camera {camera_id}")

    cap.release()

def main():
    while True:
        print("Starting snapshot capture for all cameras.")
        for index, camera_url in enumerate(camera_sources, start=1):
            capture_and_upload_snapshot(camera_url, index)

        print("Waiting for 15 minutes before next capture.")
        time.sleep(15 * 60)

if __name__ == "__main__":
    main()
```

2.1. Code Explained

1. Setting Up AWS Connection

```
s3_bucket_name = 'your-s3-bucket-name'
s3_folder_name = 'cctv_snapshots'
aws_region = 'your-aws-region'
```

This part of the code is where we define the **S3 bucket** and **region** where we want to store the snapshots from our cameras:

- **Bucket Name**: Think of an S3 bucket as a storage space in the cloud where we keep our files. We specify the bucket where each camera snapshot will be saved.
- **Folder Name**: Inside the bucket, we can create a folder called cctv_snapshots to keep everything organized.
- **AWS Region**: This is just the geographic region where your S3 storage is located, like choosing a server close to you.

2. Defining the Cameras

We list out the addresses (URLs) of the cameras that we want to connect to. These addresses are usually in a format called **RTSP**, which is commonly used for streaming video from security cameras.

```
camera_sources = [
    "rtsp://username:password@camera1_ip_address",
    "rtsp://username:password@camera2_ip_address",
]
```

Each camera has its own URL, kind of like a unique address that helps us connect to it.

3. Taking Pictures from the Cameras

The code connects to each camera, takes a snapshot (picture), saves it temporarily, and then uploads it to S3. This is all done in a function to make it reusable:

- **Connecting**: It tries to open a connection to each camera using OpenCV, a library that's great for working with images and videos.
- **Snapshot**: It takes a single picture from the camera feed. If it works, it saves the image with a timestamp (so we know exactly when the picture was taken).
- **Uploading**: After saving the snapshot on the local system, it immediately sends the file to S3 for storage.

```
def capture_and_upload_snapshot(camera_url, camera_id):

    cap = cv2.VideoCapture(camera_url)

    ...
```

This creates a unique name for each snapshot, like camera_1_20241106_143002.jpg, which helps avoid overwriting older pictures and makes it easy to find the right snapshot later.

4. Organizing Files in S3

The script places the snapshot into the cctv_snapshots folder in the specified S3 bucket. Once the file is uploaded to the cloud, the local copy is deleted to keep things tidy.

```
s3_path = f"{s3_folder_name}/{filename}"
s3.upload_file(local_path, s3_bucket_name, s3_path)
```

By moving to a real-time, cloud-based solution, we ensured that the data collection process kept pace with the project's growing requirements, ultimately providing Gartner with a more robust and responsive occupancy monitoring tool.

2.2. Assumption for running the code bit.

1. **Access to the Cameras**
   - **Network Access**: The computer running this code must be able to reach each camera. Usually, this means they're on the same local network, or the cameras are accessible online through a public IP or VPN.
   - **Correct Camera Links**: Each camera link should be set up with RTSP (Real-Time Streaming Protocol), which is common for security cameras. Think of it like needing the right "address" to connect to each camera.

2. **Camera Login Details**
   - **Usernames and Passwords**: If any cameras need a login, the code assumes you've entered those details in the list of camera links. If not, the code won't be able to access them.
   - **Reliable Connection**: The cameras should stay connected reliably. If they disconnect often or have weak signals, the code might have trouble taking consistent snapshots.

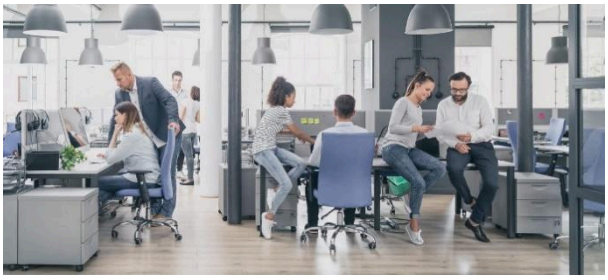
3. **AWS Credentials and Permissions**
   - **Access to AWS**: The code assumes you've set up your AWS credentials (access key and secret key) on the computer, either by using the AWS CLI or by setting environment variables.
   - **Permission to Save to S3**: These AWS credentials need permission to save (or "write") files to the S3 bucket where you want to store the snapshots.

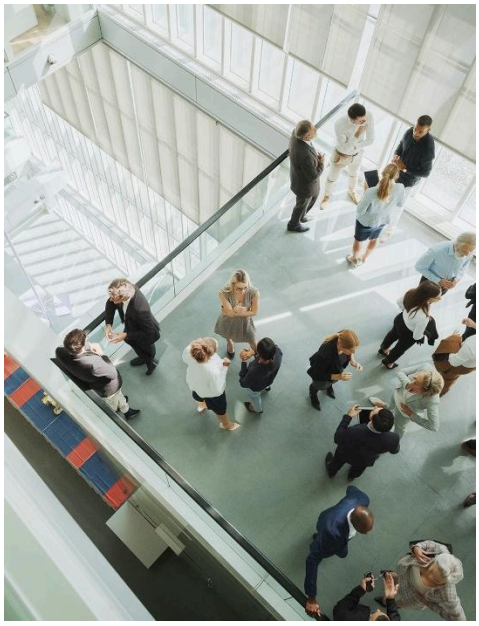
4. **S3 Bucket Exists and Is Ready**
   - **Bucket Already Created**: The code expects that the S3 bucket (your online storage space) is already set up. This code doesn't create a new bucket, so if the bucket isn't there, it won't work.
   - **Bucket in the Right Region**: The bucket should be in the region specified in the code (like us-east-1). If it's in a different region, the code might not be able to connect to it.

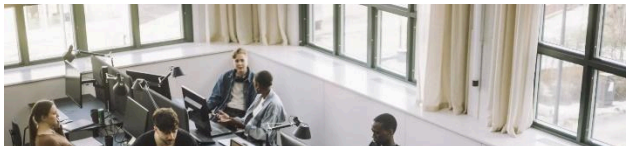
3. Collecting Images for Training

To train our detection models, we needed a good mix of real-life office images. We used popular sites like iStock to find authentic, everyday office scenes. But sometimes, we needed very specific types of images that weren't easy to find. In those cases, we turned to generative AI to

create exactly what we needed. This way, we could make sure our model was trained on a variety of images, making it better at recognizing different situations in the office.
Some of the images we used are:







4. Software Design Scope

Major Features

The major features that the product should support are:

- **Pose Detection**: Use YOLOv8 to identify humans and detect pose keypoints (eyes, nose, etc.) for enhanced analysis of group interactions.
- **Group Detection and Analysis**: Identify groups based on proximity, pose keypoints, and gaze direction to detect if two people are interacting.
- **Gaze Detection Algorithm**: Calculate if two people are looking in the same direction using pose keypoints, vector mathematics, and proximity analysis.
- **Data Storage**: Store pose keypoints, detection timestamps, and group analysis outputs for future insights.
- **Dashboard Visualization**: Present key insights, group detection results, and gaze detection information in an interactive dashboard via Power BI.

Design Constraints

The application must be responsive to ensure compatibility with multiple devices and platforms, particularly when viewing outputs on Power BI. All features must be user-friendly and intuitive. The color scheme must meet UI design standards for optimal readability and accessibility, with special attention to color contrast.

Some Limitations

The application is hosted on a free cloud environment, limiting data processing speeds, storage, and potential access times. Google Colab's free usage may limit processing time. Additionally, without HTTPS, the web application may be vulnerable to HTTP attacks.

Possible Future Enhancements

- **Advanced Real-Time Processing**: Improved gaze tracking with real-time visualization.
- **Enhanced Dashboard Capabilities**: Customizable graph displays in Power BI.
- **Automated Data Backup**: Integration with cloud storage for automatic data backup.
- **SSL Certificate for Security**: Ensure HTTPS protection to reduce security vulnerabilities.
- **Optimized Server Architecture**: For faster data processing and load times.

Reference Documents

Existing Software Documentation

Existing documentation includes project requirements provided by the client and a summary of the proposed system.

System Documentation

Since this project is standalone, no system integration documents are applicable.

Vendor Documentation

Machine Learning and Detection Libraries

- **Ultralytics YOLOv8**: Provides the YOLOv8 model for human and pose detection tasks, identifying people, key points, and interactions for advanced analytics.
    - Ultralytics YOLOv8 Documentation
- **Roboflow**: Assists with dataset management, preprocessing, and annotation for optimizing model training and inference accuracy.
    - Roboflow Documentation
- **Python**: Main programming language for model development, data processing, and mathematical calculations for gaze detection.
    - Python Documentation

Development Environment

- **Google Colab**: Hosted cloud environment used to develop and test models, process data, and manage code execution.
    - Google Colab

Visualization Tools

- **Power BI**: Used for creating dashboards, heatmaps, and interactive visualizations for data analysis.
    - Power BI Documentation

Data Storage Solutions

- **SQLite**: Local database to store model outputs, pose keypoints, group detection results, and timestamps efficiently.
    - SQLite Documentation

Understanding YOLO V8 for Real-Time Object Detection

1. Introduction

Overview

YOLOv8 (You Only Look Once, version 8) is a state-of-the-art real-time object detection model known for its speed and accuracy. Originally developed by Joseph Redmon, YOLO's early versions introduced a new approach to object detection by processing the entire image in a single pass, unlike other models that scanned images in sections. This architecture made YOLO one of the fastest and most efficient object detectors, allowing it to excel in real-time applications.

2. Overview of YOLO (You Only Look Once) Framework

The Evolution of YOLO

Since its inception, YOLO has evolved significantly, with each version enhancing accuracy, speed, and usability:

- **YOLOv1 - v3**: Early versions were foundational, introducing YOLO's unique one-pass detection and achieving high speeds.
- **YOLOv4 and v5**: These versions focused on improving accuracy and adapting to smaller object detection.
- **YOLOv7**: Enhanced real-time capabilities with architectural adjustments.
- **YOLOv8**: The latest version, optimized for accuracy, flexibility, and user-friendliness, making it easier to fine-tune and adapt for custom applications.

Single-Pass Detection

YOLO processes the entire image in a single forward pass, breaking it into grids and predicting bounding boxes and class probabilities for each grid cell simultaneously. This one-pass approach is faster than traditional multi-stage detectors like Faster R-CNN, making YOLO ideal for high-speed applications.

3. Why YOLOv8?

Key Reasons for Choosing YOLOv8

- **Enhanced Speed and Accuracy**: YOLOv8 combines the speed of earlier versions with improved accuracy, suitable for real-time applications like surveillance and group detection.
- **Ease of Use and Flexibility**: YOLOv8's modular architecture makes it user-friendly and easy to customize, enabling developers to quickly adapt it for specific tasks such as human pose detection.

- **Multi-Scale Object Detection**: YOLOv8 improves the ability to detect small and large objects simultaneously, which is essential for human interaction analysis.
- **Training on Custom Data**: YOLOv8 is optimized for training on custom datasets, allowing us to fine-tune the model for specialized scenarios, such as identifying key points and analyzing poses in office environments.
- **Integration with Other Libraries**: YOLOv8's compatibility with PyTorch and Ultralytics makes it easy to integrate with existing deep learning workflows.

Why YOLOv8 Over Alternatives?

Compared to alternatives like Faster R-CNN and SSD, YOLOv8 is particularly suited for real-time, dynamic environments. While Faster R-CNN offers high accuracy, it is slower and less efficient for live feeds. SSD is faster but generally less accurate than YOLOv8, especially in detecting small or overlapping objects.

4. How YOLOv8 Works

Model Architecture

YOLOv8 divides an image into a grid of cells, and for each cell, it predicts bounding boxes and associated confidence scores. The architecture can be broken down as follows:

- **Grid Cell Assignments**: Each grid cell is responsible for detecting objects within its bounds.
- **Bounding Box Predictions**: YOLOv8 predicts multiple bounding boxes per cell and refines them based on anchor boxes, making it more adaptive to varying object sizes.
- **Class Probabilities**: Alongside bounding boxes, YOLOv8 calculates class probabilities to identify detected objects.
- **Non-Maximum Suppression (NMS)**: After detecting multiple bounding boxes, NMS eliminates redundant boxes, leaving only the most confident predictions.

Key Components

- **Backbone Network**: YOLOv8 uses a deep convolutional network that extracts key features from input images.
- **Neck and Head**: The neck layers aggregate features, while the head layers are responsible for the final object detection and classification.
- **Loss Function**: YOLOv8 optimizes for both classification and localization, balancing accuracy with detection speed.

5. Benefits of YOLOv8 for Human Interaction Analysis

Real-Time Processing

YOLOv8's speed makes it ideal for real-time analysis of office environments. Its rapid inference capability means that live video feeds can be processed efficiently without sacrificing accuracy.

Pose Estimation Capabilities

YOLOv8 is compatible with pose estimation extensions, which makes it suitable for analyzing human interactions based on body posture and gaze direction. By detecting keypoints such as eyes and nose, YOLOv8 facilitates complex interaction analysis, especially useful in determining if people are looking at each other.

Reduced False Positives

YOLOv8's improved bounding box predictions and anchor-based detection minimize false positives. For example, in group detection tasks, YOLOv8 helps reduce the likelihood of mistakenly classifying two people as interacting just because they're close to each other.

Enhanced Data Availability

YOLOv8 can operate with pre-trained models or be fine-tuned on a custom dataset, making it flexible for various applications. Pre-trained weights allow us to start with a high-performing base, while custom training refines the model for specific detection tasks like distinguishing between groups and individuals.

6. Training YOLOv8 for Custom Use Cases

Data Preparation

For training YOLOv8 on custom data, images need to be annotated with bounding boxes or keypoints based on the intended detection. Popular tools like LabelIng and VGG Image Annotator (VIA) are suitable for generating these annotations.

Transfer Learning

YOLOv8 supports transfer learning, allowing the model to start from pre-trained weights. This approach accelerates training by leveraging existing knowledge from similar datasets.

Training Parameters

Key parameters to adjust during training:

- **Learning Rate**: Adjusts the rate of convergence to prevent overfitting.
- **Batch Size**: Larger batch sizes enhance gradient stability.
- **Epochs**: Determining the optimal number of epochs ensures a balanced model that isn't overfit.

Fine-Tuning Techniques

Once trained, fine-tuning further optimized YOLOv8 for specific metrics like accuracy and inference speed. Adjusting confidence thresholds and non-maximum suppression values ensures that YOLOv8 remains precise in group detection tasks.

7. Challenges and Best Practices

Challenges

- **Occlusion and Crowding**: When people are close together or partially occluded, distinguishing individuals becomes challenging.
- **Lighting and Environmental Changes**: Variations in lighting can impact detection performance, especially in real-time applications.
- **Dynamic Backgrounds**: Complex or moving backgrounds may introduce noise and false positives.

Best Practices

- **Confidence Threshold Tuning**: Lowering the confidence threshold can help detect smaller objects, while higher thresholds reduce noise.
- **Data Augmentation**: Techniques like rotation, flipping, and brightness adjustments make the model more robust to varied real-world conditions.
- **Model Pruning**: For environments with limited computational power, pruning non-essential layers can enhance YOLOv8's efficiency.

8. Further Reading

Educational Resources

- **Object Detection for Beginners**: A basic guide on object detection principles.
- **YOLO Algorithm Explained**: Detailed breakdowns of YOLO's inner workings.
- **Pose Estimation with YOLOv8**: Tutorials and documentation on keypoint detection.

Technical Documentation

- **Ultralytics YOLOv8 GitHub Repository**: YOLOv8 GitHub – Complete source code, examples, and pre-trained models.
- **YOLOv8 Documentation**: YOLOv8 Docs – Comprehensive technical details on using YOLOv8 with Ultralytics.
- **PyTorch and YOLO Integration Guide**: Resources for integrating YOLOv8 with PyTorch for custom model training.

Academic References

- **Research on YOLOv8 and Real-Time Detection**: Articles on the application of YOLOv8 in live environments.
- **Pose Estimation with YOLO**: Research papers on incorporating pose detection in YOLO models.

Introduction to Group Detection

Purpose

The group detection module aims to identify and analyze clusters of people within a monitored space based on their spatial relationships and visual characteristics. By detecting groups, this module provides insights into patterns of occupancy, interactions, and overall crowd dynamics. This analysis can inform space management, optimize layouts, and enhance safety measures by showing how people naturally congregate or interact within different areas.

Using bounding box information from object detection algorithms, the module classifies people into groups based on proximity and size comparisons of bounding boxes. This methodology helps ensure that only individuals in close physical proximity are grouped together, reflecting real-world clustering patterns.

Overview of Final Approaches

To accurately detect groups, multiple methods were developed and refined, with two primary approaches proving effective:

- **Bounding Box Proximity**: Using the centroids of bounding boxes, distances between detected individuals are calculated. If two bounding boxes fall within a set distance threshold, they are considered part of the same group. This approach works well in straightforward scenarios where individuals are in proximity and at similar distances from the camera.
- **Bounding Box Area Ratio Comparison**: In environments where proximity alone may not reflect actual grouping—especially in scenes with varying distances from the camera—bounding box areas are compared. If two people are close in centroid distance but have significantly different bounding box areas, they are less likely to be grouped. This method mitigates grouping errors caused by perspective distortion or varying distances in wide-angle or fisheye camera views.
- **DPTZ Adjustments for Fisheye Images**: Given that fisheye cameras introduce considerable image distortion, direct use of the fisheye output made accurate group detection challenging. The Digital Pan-Tilt-Zoom (DPTZ) adjustment is a workaround that converts fisheye images into a standard rectangular format, reducing distortions and improving the effectiveness of both proximity and area-based group detection.

Explored Approaches and Limitations

In the process of refining the group detection system, several approaches were explored but ultimately set aside due to their limitations in this application:

- **Clustering of Bounding Boxes**: Unsupervised learning algorithms like DBSCAN and K-Means were initially considered. However, given that each image had only 8-10 data points (bounding boxes), these clustering algorithms were overkill. Additionally, their effectiveness was limited by the low number of data points, which did not justify their computational cost or complexity.
- **Density-Based Heatmaps**: While density heatmaps helped visualize people's locations, they lacked the specificity required to define groups. Heatmaps show general areas of higher density but do not distinguish between individual clusters or enable clear group separation.
- **Intersection Over Union (IoU)**: IoU was used to assess overlap between bounding boxes, aiming to identify group members based on overlap. Although IoU worked in some scenarios, it was inconsistent, as individuals in the same group are not always close enough to have overlapping bounding boxes. Thus, IoU alone could not reliably classify groups.
- **Bounding Box Area Comparison Alone**: Initially, comparing bounding box areas was considered a standalone method to gauge distance and clustering. However, this approach proved overly specific and required excessive hyperparameter tuning for different camera angles and settings. Consequently, bounding box area comparison was incorporated as a supplementary check rather than the primary method.

System Requirements

Software and Libraries

The group detection system relies on specific software and libraries to perform person detection, group classification, and handle fisheye camera adjustments:

- **Python 3.8+**: The primary language for implementing the solution.
- **Libraries**:
    - **OpenCV (Version 4.5.5+)**: For image processing, visualization, and drawing bounding boxes.
    - **YOLOv8 (Ultralytics)**: Used for optimized person detection, with pre-trained weights suited for the task.
    - **SciPy (Version 1.8.0+)**: Calculates distances between bounding box centers for group classification based on centroid proximity.
    - **NetworkX (Version 2.6+)**: Used to create and analyze graphs for grouping based on detected proximity.
    - **NumPy (Version 1.21+)**: Supports array handling and efficient bounding box coordinate processing.
    - **Matplotlib (Optional, Version 3.4+)**: For data visualization, beneficial for system testing or presentation.
- **DPTZ Interface Dependencies**:

    - For fisheye cameras like the Meraki MV32, DPTZ (Digital Pan-Tilt-Zoom) adjustments allow the fisheye view to be converted to a standard rectangular view, reducing distortion. Access to the camera's software interface or SDK may be required for these transformations.
- **Operating System**: The solution is compatible with Windows, macOS, and Linux, provided the system supports the above libraries.

Hardware and Platform Recommendations

- **Testing Environment**: So far, development and testing have been conducted on Google Colab, a platform suited for rapid prototyping. Colab's GPU resources have been effective for testing purposes.
- **Production Deployment**: For a live, scalable solution, deploying to a cloud platform like **AWS** is recommended. This would enable:
    - **Real-time Processing**: Faster response times for continuous video streams.
    - **Scalability**: Easily scale resources based on usage and video load.
    - **Reliability**: High availability and minimal downtime, essential for real-world applications in dynamic environments.
- **Camera Positioning**: Positioning the camera at a 45-degree angle is preferred for optimal group detection accuracy, as it balances the perspective, minimizing distortion due to varying depths.

Configuration and Setup

1. Parameter Tuning

To achieve accurate group detection, several parameters are adjustable to tailor the system's performance to specific environments and camera angles. Here are the key parameters and recommended default values:

- **Proximity Threshold (Centroid Distance)**:
    - **Default Value**: 150-300 pixels
    - **Purpose**: Defines the maximum allowable distance between two bounding box centers for them to be considered as part of the same group. Lower values group people only when they are very close, while higher values expand grouping to include individuals farther apart.
    - **Effect**: Adjusting this threshold helps in managing the balance between grouping accuracy and flexibility. Higher thresholds can lead to incorrect grouping in crowded areas, while lower values may prevent valid groups from being identified.
- **Bounding Box Area Ratio Threshold**:
    - **Default Value**: 0.5 (or 50%)

    - **Purpose**: Determines whether to group two people based on a comparison of their bounding box areas, filtering out individuals far from the camera.
    - **Effect**: Lower thresholds increase the likelihood of individuals of varying distances being grouped together, while higher thresholds enforce stricter grouping based on similar bounding box sizes.
- **Minimum and Maximum Grouping Distances**:
    - **Default Values**:
        - Minimum Distance: 50 pixels
        - Maximum Distance: 150 pixels
    - **Purpose**: Ensures that only people within these distances are considered for grouping, preventing erroneous grouping between close-up and distant individuals.
    - **Effect**: These values can be adjusted to prevent overlap between groups in environments with different crowd densities or camera setups.

2. DPTZ Interface Setup

To minimize distortion and improve detection, switching the fisheye camera (e.g., Meraki MV32) to DPTZ mode is essential. This process converts the fisheye view to a standard, rectangular view:

- **Accessing DPTZ Mode**:
    - Login to the camera interface using the provided credentials (or the network's IP address).
    - Navigate to the camera settings panel.
    - Enable **DPTZ Mode** or **Rectilinear Projection** (based on the camera model's options).
    - Adjust the **Field of View (FOV)** and **Pan-Tilt-Zoom** features if necessary, to capture the desired monitoring area.
- **Recommended Settings**:
    - **Field of View**: 90-120 degrees for balanced visibility.
    - **Zoom**: Minimal zoom to capture broader areas without affecting detection range.

Component Workflow

Detection Process

The group detection process involves a sequence of stages, each critical for accurately identifying individuals and groups in the camera's field of view. Here is a detailed breakdown of each component in the workflow:

1. **Object Detection with YOLO**
    - **Purpose**: YOLO (You Only Look Once) is employed to detect individuals in each frame or image, serving as the initial step for identifying people within the scene.
    - **Process**:

        - YOLO processes the image frame and identifies persons based on pre-trained deep learning models, which recognize people by detecting distinct visual features.
        - Each detected person is assigned a **bounding box** with a unique identifier and coordinates for the box edges.
        - **Output**: This stage generates a set of bounding boxes for each detected person, including data on bounding box positions and sizes.
2. **Bounding Box Proximity and Ratio Comparison**
    - **Purpose**: This stage groups detected individuals based on spatial proximity and size similarity, enhancing group detection accuracy.
    - **Process**:
        - **Centroid Calculation**: For each bounding box, the centroid (center point) is calculated.
        - **Bounding Box Proximity Check**:
            - The Euclidean distance between centroids is measured, and pairs of bounding boxes are flagged as potential groups if they are within a specified **proximity threshold**.
        - **Bounding Box Ratio Comparison**:
            - Bounding box area ratios are calculated between pairs of nearby bounding boxes.
            - If the ratio falls within a specified range (indicating similar distance from the camera), the boxes are likely part of the same group. Significant differences in area indicate that one person is farther from the camera, so they are excluded from the grouping.
        - **Graph-Based Group Formation**:
            - Using **NetworkX**, a graph is constructed where each detected individual is a node, and edges represent connections based on proximity and area criteria.
            - Connected nodes are then clustered into groups.
        - **Output**: This stage outputs groups of connected individuals based on proximity and area similarity, representing logical groups within the scene.
3. **DPTZ for Fisheye Correction**
    - **Purpose**: Fisheye cameras introduce significant image distortion, which can affect object detection accuracy. Switching to **DPTZ (Digital Pan-Tilt-Zoom)** mode helps mitigate this issue by offering a standard rectangular view.
    - **Process**:
        - Using the fisheye camera's built-in settings, the DPTZ mode is enabled, which adjusts the camera feed to a **rectilinear** perspective.
        - This removes distortion from the edges of the fisheye image, normalizing bounding box shapes and sizes, and ensuring that people across various angles are detected with higher accuracy.
    - **Output**: The DPTZ view corrects image distortions, providing a reliable foundation for YOLO to detect individuals more accurately and facilitating better group detection by standardizing bounding box properties.

Common Issues and Troubleshooting

Detection Errors

Detection errors, such as misclassified groups, can often be attributed to improper tuning of key parameters. Here are some recommendations to help fine-tune the settings:

- **Adjusting Proximity Threshold**:
    - **Issue**: If groups that should be detected together are classified separately, the **proximity threshold** may be set too low.
    - **Solution**: Incrementally increase the proximity threshold until groups that are close in real-life scenarios are correctly classified. However, be cautious of setting it too high, as this may merge distinct groups.
    - **Recommendation**: Test the threshold values in varied environments and angles to ensure robust accuracy across different scenarios.
- **Bounding Box Area Ratio**:
    - **Issue**: Groups are incorrectly merged due to varying sizes of bounding boxes, especially in cases where individuals are at different distances from the camera.
    - **Solution**: Adjust the **bounding box area ratio threshold** to better distinguish between individuals at different distances. Lowering the threshold can help separate individuals who are farther apart in depth, while increasing it may work better for groups at similar distances.
    - **Recommendation**: Evaluate the bounding box area ratio values for each environment or camera position to refine grouping accuracy.

False Positives/Negatives

False positives or negatives can occur when individuals are either incorrectly grouped or missed as a group. Here are solutions to minimize these errors:

- **False Positives (Over-Grouping)**:
    - **Issue**: People who aren't part of the same group are being classified as one group, often due to similar bounding box sizes or close proximity in the image.
    - **Solution**:
        - Reduce the **bounding box area ratio threshold** to tighten grouping criteria.
        - Apply additional **filters for edge cases** where individuals may appear close together but aren't logically grouped (e.g., by setting constraints based on camera angle or specific regions in the image where false positives frequently occur).
    - **Recommendation**: Run tests to identify areas where over-grouping happens and adjust the ratio or proximity threshold accordingly.
- **False Negatives (Under-Grouping)**:
    - **Issue**: People who should be grouped together are being missed, often due to bounding box sizes or centroid distances slightly exceeding the threshold.

    - **Solution**:
        - Increase the **proximity threshold** or **expand the bounding box area ratio range** slightly to allow more flexibility in grouping, especially for individuals who might be standing farther apart but are in the same group.
    - **Recommendation**: Adjust these parameters incrementally to balance grouping accuracy while minimizing the inclusion of unrelated individuals.

Performance Optimization

Processing Speed

To achieve an optimal balance between accuracy and processing speed, here are some tips:

- **YOLO Confidence Threshold**:
    - Adjust the confidence threshold to filter out low-confidence detections, reducing processing time and focusing only on higher-confidence detections.
    - **Recommendation**: Start with a threshold of around 0.4 and test in your specific environment to find the optimal level.
- **Resizing Input Images**:
    - Resizing images can significantly improve processing speed by reducing the computational load. Lower resolution may, however, reduce detection accuracy for smaller or distant objects.
    - **Recommendation**: Use a smaller image size for rapid detection needs, with testing to confirm if accuracy remains acceptable.

Model Updates

To keep group detection performance current, consider these model update practices:

- **Integrating New YOLO Models**:
    - YOLO's architecture continually evolves, and new models may offer better accuracy or efficiency. Integrate updates by replacing the model weights in the code and adjusting the detection parameters as necessary.
    - **Recommendation**: Check YOLO and other object detection libraries periodically for updates or lightweight models that suit your deployment environment.
- **Alternative Object Detection Models**:
    - Explore models like EfficientDet or SSD if they offer comparable accuracy with improved speed or better fit specific camera and scene characteristics.
    - **Recommendation**: Test and evaluate alternate models in your setup, especially if they provide computational efficiency or detection improvements.

Future Maintenance and Updates

Parameter Updates

- **Monitoring and Adjusting Parameters**:
    - Over time, camera placement or scene layouts may change, affecting group detection. Regularly monitor and update key parameters (e.g., proximity threshold, bounding box ratio) to accommodate these changes.
    - **Recommendation**: Establish a regular review schedule to ensure parameters are optimally set based on new conditions.

Code Modifications

- **Adapting Key Sections**:
    - Key sections of the code, particularly where detection logic and grouping criteria are set, should be well-documented and modular for easy modification.
    - **Recommendation**: Modularize code for bounding box detection, proximity, and ratio comparisons to make future enhancements, such as integration with additional sensors or analysis modules, more seamless.

Documentation

- **Maintaining an Update Log**:
    - Record all parameter changes, detection model updates, or configuration modifications in a log, helping future users quickly understand the system's evolution.
    - **Recommendation**: Include notes on the rationale for each change, parameter values, and observed effects on performance or accuracy.

Gaze Detection System: A Comprehensive Guide

Understanding Human Interaction Analysis

1. Introduction

Overview

The gaze detection system is an innovative approach to understanding human interactions in images and video footage. By analyzing where people are looking, the system can better determine whether individuals are actually interacting with each other, leading to more accurate group identification.

Why Gaze Matters

Traditional group detection systems often rely solely on physical proximity to identify groups. However, in crowded spaces, people might be physically close without actually interacting. Our gaze detection system addresses this limitation by considering where people are looking, making the group detection more intelligent and human-like.

2. Understanding Pose Detection

What is Pose Detection?

Pose detection is like creating a digital skeleton of a person in an image. The system identifies key points on the human body, similar to how artists use reference points when drawing figures. For our gaze detection system, we focus primarily on three crucial points:

- The nose
- The left eye
- The right eye

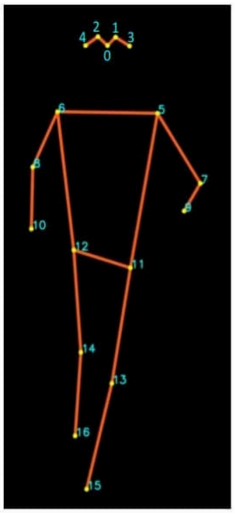
The YOLOv8 System

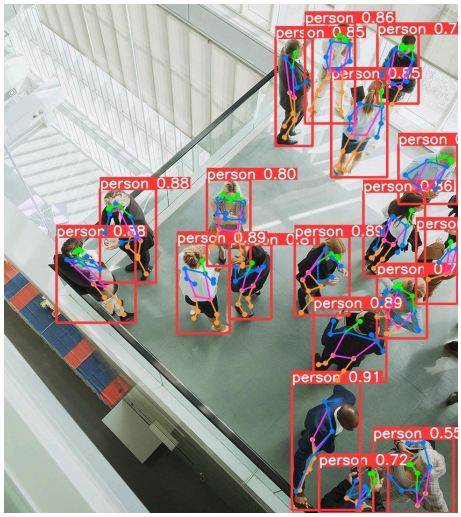
We utilize YOLOv8 (You Only Look Once, version 8) for pose detection. YOLOv8 is a powerful technology capable of:

- Scanning images quickly to detect multiple people
- Identifying key body points in real-time
- Working efficiently and accurately even in dynamic environments

These capabilities make YOLOv8 an ideal solution for detecting people and their body points, which are essential for gaze analysis.

Index	Key point
0	Nose
1	Left-eye
2	Right-eye
3	Left-ear
4	Right-ear
5	Left-shoulder
6	Right-shoulder
7	Left-elbow
8	Right-elbow
9	Left-wrist
10	Right-wrist
11	Left-hip
12	Right-hip
13	Left-knee
14	Right-knee
15	Left-ankle
16	Right-ankle





3. How Gaze Detection Works

The Basic Principle

Our gaze detection works similar to drawing an invisible line from a person's eyes in the direction they're looking. Here's how:

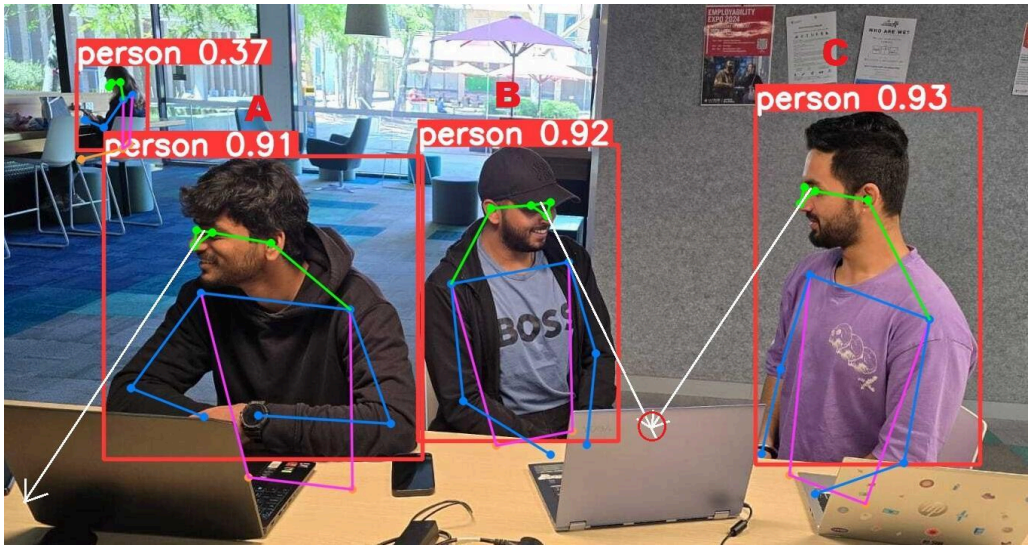
1. **Starting Point**
    - The system finds the middle point between a person's eyes
    - If only one eye is visible, it uses that eye's position
    - This becomes the origin of our "gaze line"
2. **Direction Finding**
    - The nose position serves as a direction indicator
    - The system draws an invisible line from the eye midpoint through the nose
    - This line represents the person's general direction of gaze
3. **Interaction Analysis**
    - The system checks if two people's gaze lines intersect
    - It measures the angle between their gazes
    - This helps determine if they're likely interacting

Understanding Vector Mathematics

While the underlying math is complex, the concept is simple: imagine two people looking at each other. Their gaze lines would cross at a point between them. If they're not looking at each other, these lines either won't meet or will meet at an odd angle.

A raw is drawn from the midpoint of the eyes, passing through the nose.
If it intersects under the threshold value of angle, then 2 people are considered in a group and an edge is drawn b/w nodes.

In the image we can see, B and C's rays are intersecting. But A's ray is not intersecting with B, so no edge will be drawn between their nodes.



4. Group Classification Process

Multi-Step Verification

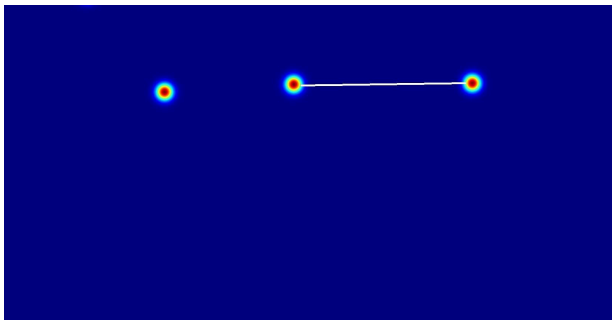
The system uses three main criteria to determine if people are in a group:

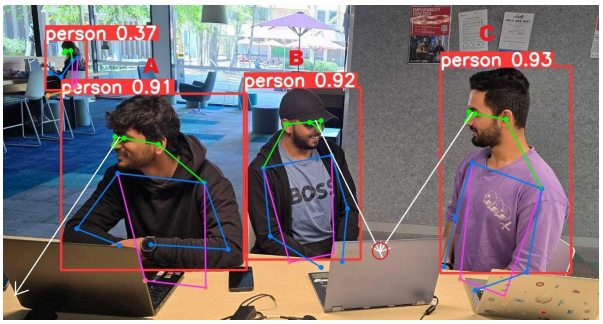
1. **Distance Check**
    - Are the people within a reasonable distance?
    - Not too close (which might be coincidental in crowds)
    - Not too far (beyond normal interaction distance)
2. **Size Comparison**
    - Are the people similar in size in the image?
    - This helps account for perspective and distance from the camera
    - Prevents false grouping of people at different distances
3. **Gaze Analysis**
    - Are they looking in compatible directions?
    - Do their gaze lines intersect?
    - Is the angle between their gazes reasonable?

Group Formation Rules

The system follows some key rules when forming groups:

- People must satisfy all three criteria to be considered a group
- Groups can include more than two people
- Each person can only belong to one group
- Single individuals are not marked as groups





Even though A and B are much closer, as compared to B and C; but since they are not looking at each other, they will not be considered in a group.

5. Technical Concepts Made Simple

Understanding Thresholds

The system uses several thresholds to make decisions:

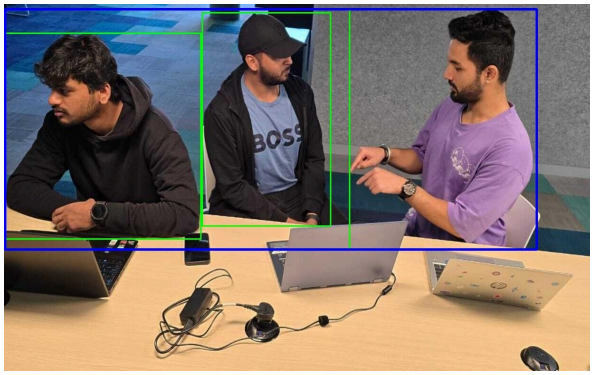
- **Angle Threshold**: Usually 90 degrees - if people are looking at angles greater than this, they're probably not interacting
- **Distance Thresholds**: Minimum and maximum distances that make sense for human interaction
- **Size Ratio**: Helps adjust for camera perspective, comparing the sizes of detected individuals to determine if they are on the same plane.

Visualization

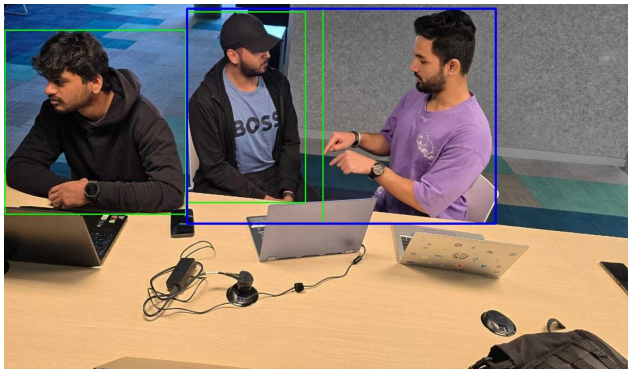
The system provides visual representations to aid understanding, including:

- **Individual Detection**: Showing detected people and their pose points.
- **Group Boundaries**: Highlighting defined groups.
- **Gaze Directions**: Visualizing the gaze lines.
- **Heat Maps**: Indicating crowd density and interaction hotspots.

Using pose helps reduce false positives in identifying groups.



Group Detection without using pose



Group Detection without using pose

6. Common Challenges and Solutions

Typical Challenges

1. **Partial Visibility**
    - When faces are partially hidden
    - When people are turned sideways
    - In crowded scenes
2. **Environmental Factors**
    - Poor lighting conditions
    - Busy backgrounds
    - Motion blur in videos

Solutions Implemented

- Robust error handling for missing facial points
- Alternative calculation methods when ideal data isn't available
- Confidence scoring to ensure reliability

7. Future Developments

Potential Improvements

- **Emotion Recognition**: Adding an emotional layer to identify context in interactions.
- **Adaptive Thresholds**: Dynamic adjustment based on different environments and cultural contexts.
- **Enhanced Group Dynamics**: Supporting complex group interactions with more advanced modeling.

Research Directions

- **Machine Learning Integration**: Using machine learning to adapt thresholding and detect complex patterns.
- **Cultural Interaction Patterns**: Studying variations in gaze and group behavior across cultures.
- **Behavioral Analysis Integration**: Combining gaze and pose data for richer behavioral insights.

8. Further Reading

Educational Resources

- **Introduction to Computer Vision**: Basic concepts of image processing and object detection.
- **Understanding Human Behavior Analysis**: Insights into human interactions and body language.
- **Modern Pose Estimation Techniques**: Background on technologies like YOLOv8 and pose estimation methods.

Academic References

- Research articles on **YOLO** for object detection.
- **Human Interaction Analysis** papers for understanding group behavior and gaze analysis.
- Studies on **Crowd Dynamics** for insights on real-world applications of gaze-based interaction detection.

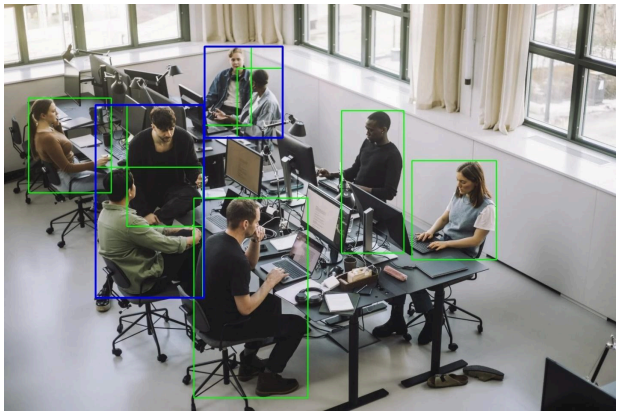
Online Resources

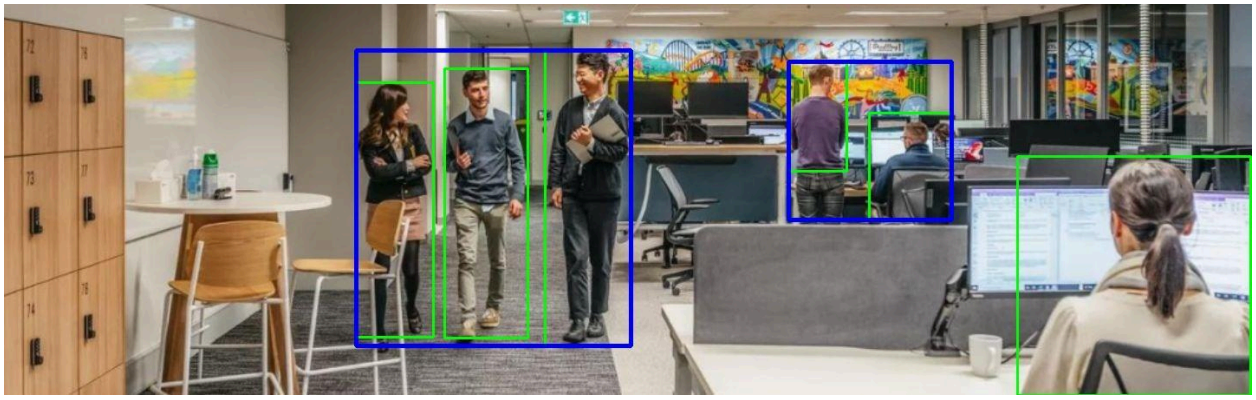
- **Ultralytics YOLOv8 Documentation**: Comprehensive guide to YOLOv8.
- **Computer Vision Tutorials**: Practical resources on computer vision and pose estimation.
- **Human Behavior Analysis Guides**: Understanding human interactions and social group dynamics.

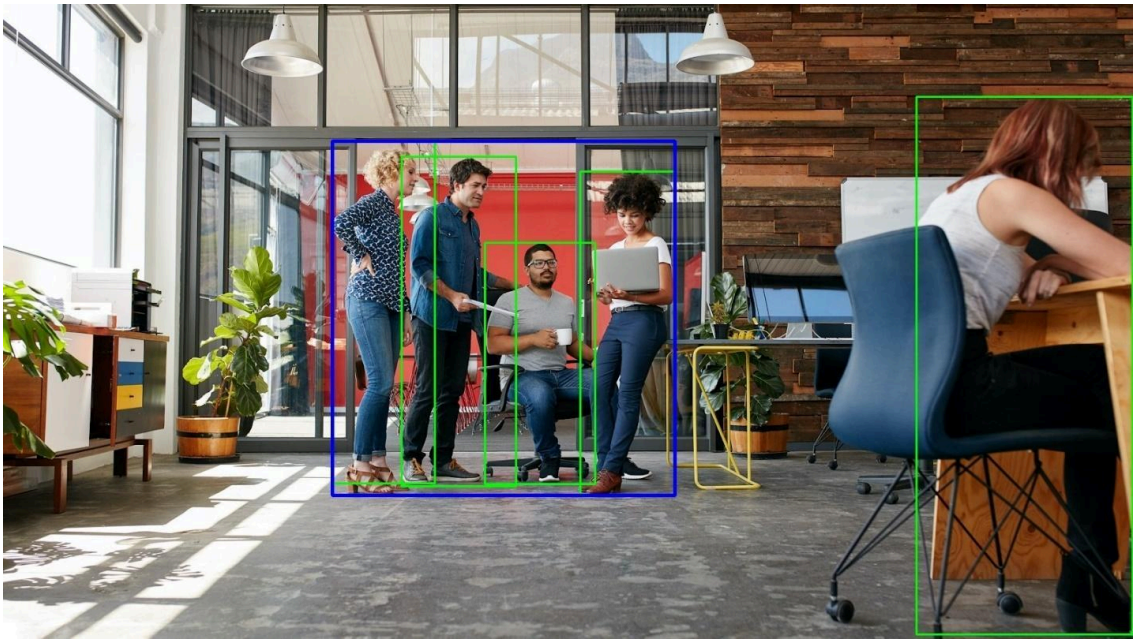
This guide is designed to serve as a comprehensive introduction to the gaze detection system, giving new developers and researchers the context and foundational knowledge to dive into the technical documentation. For detailed implementation steps or code examples, please refer to the specific technical resources or consult the development team.

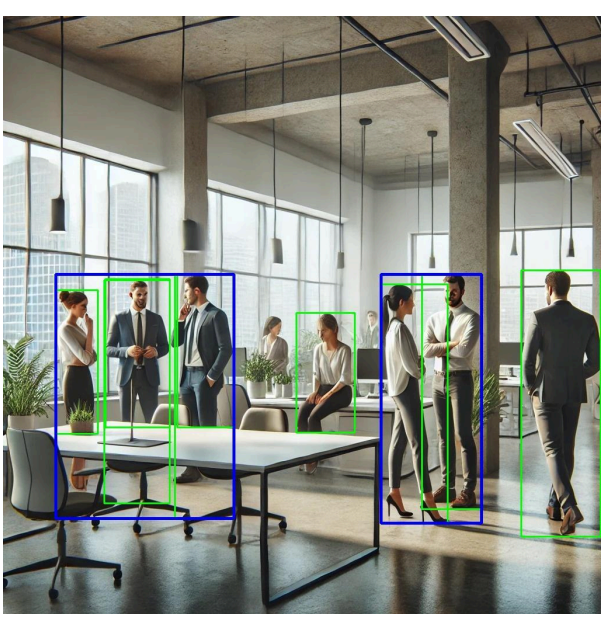
Final model outputs

The following images are of group detection only using the bounding box proximity feature.

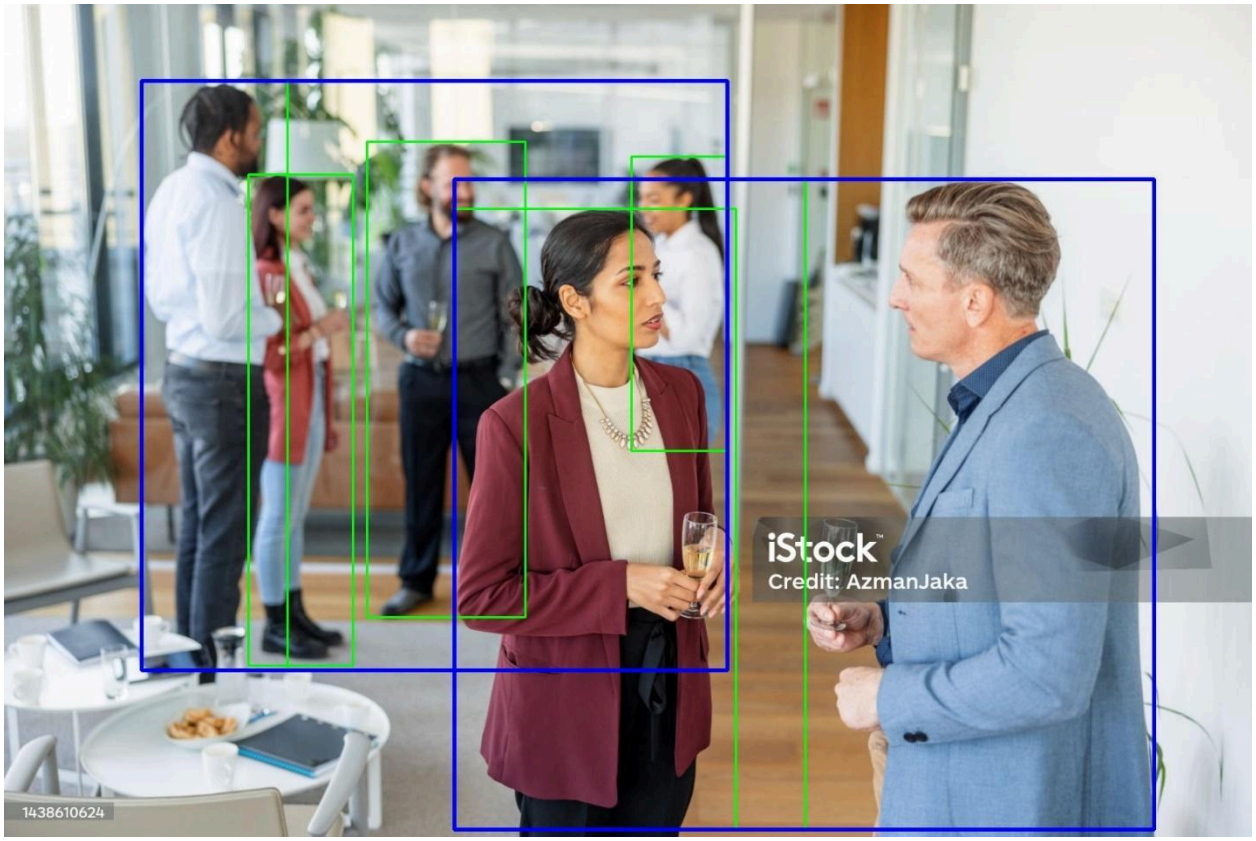


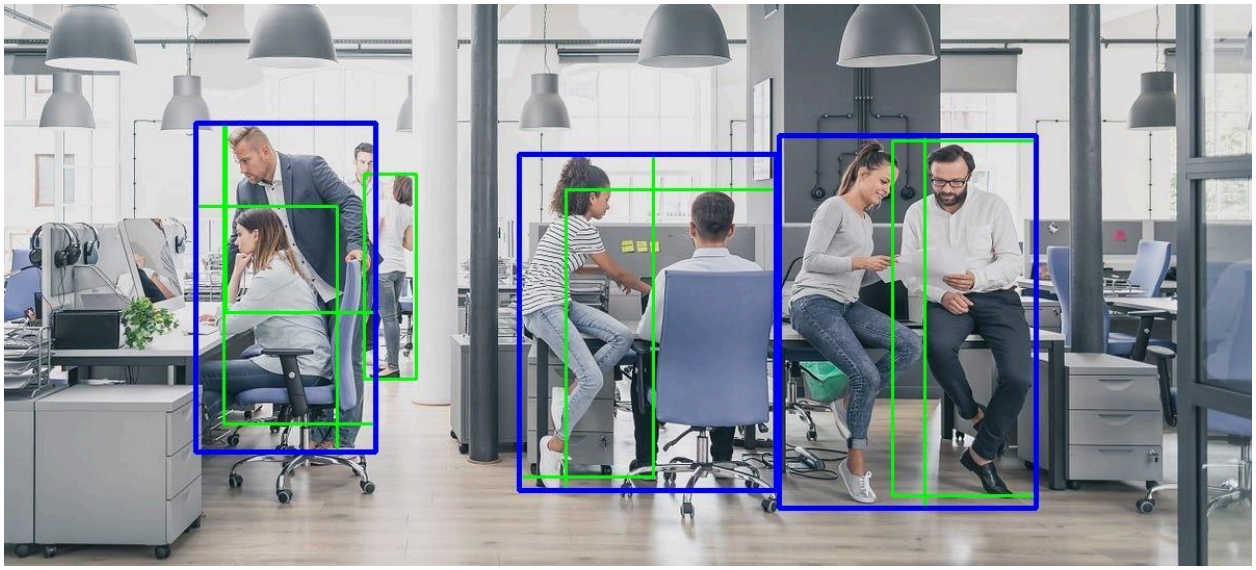


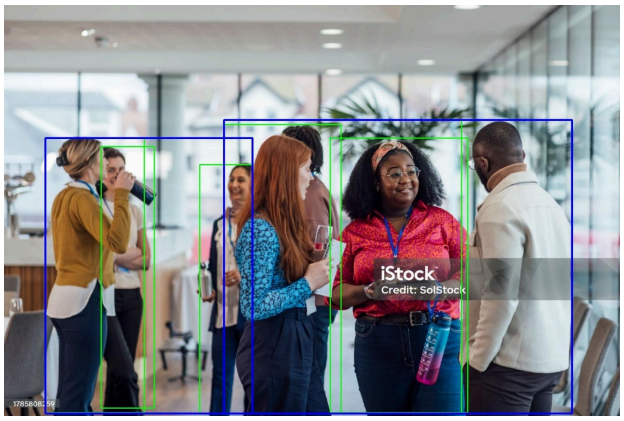


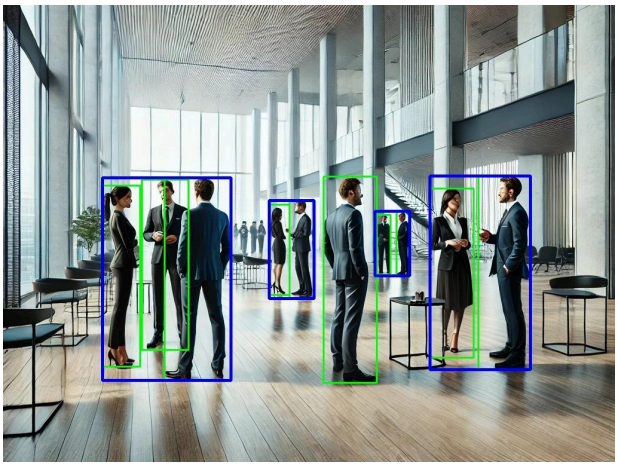


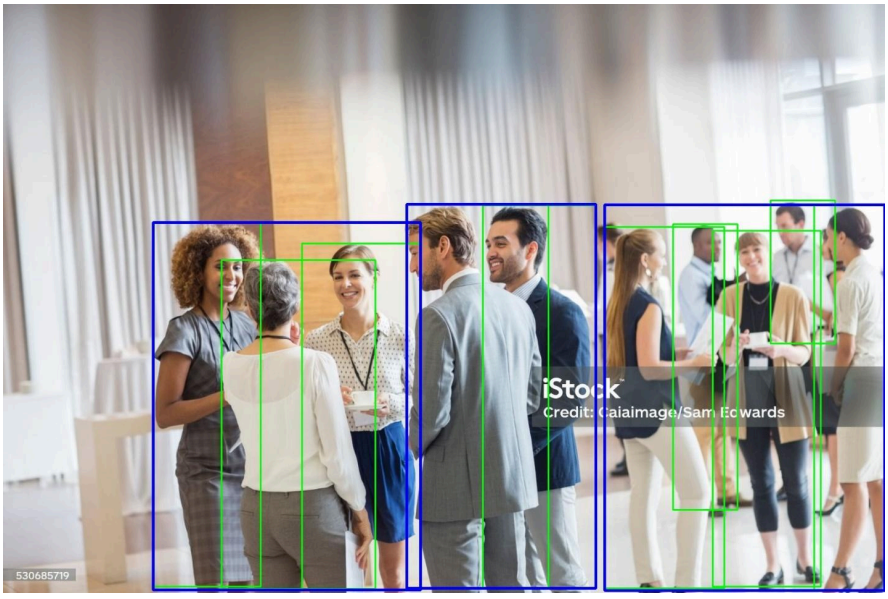
The following images are of group detection by using both bounding box proximity and bounding box area ratio comparison features.

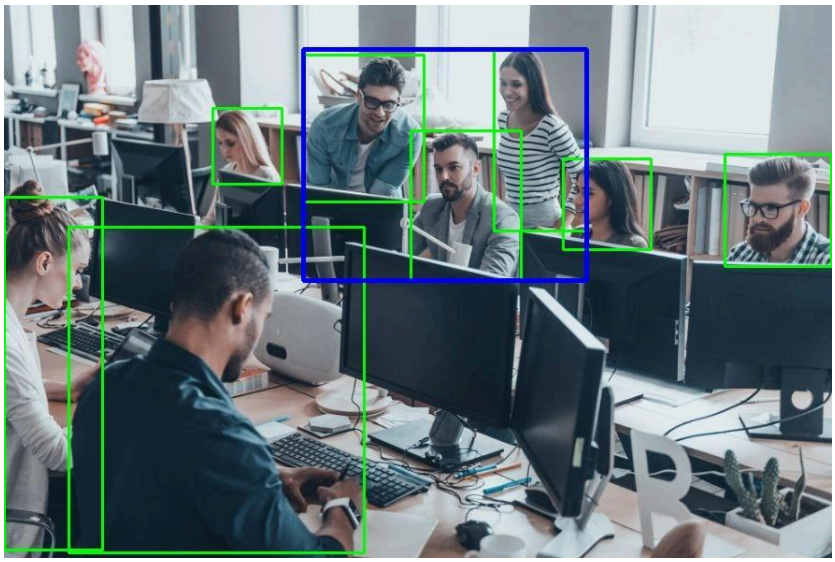




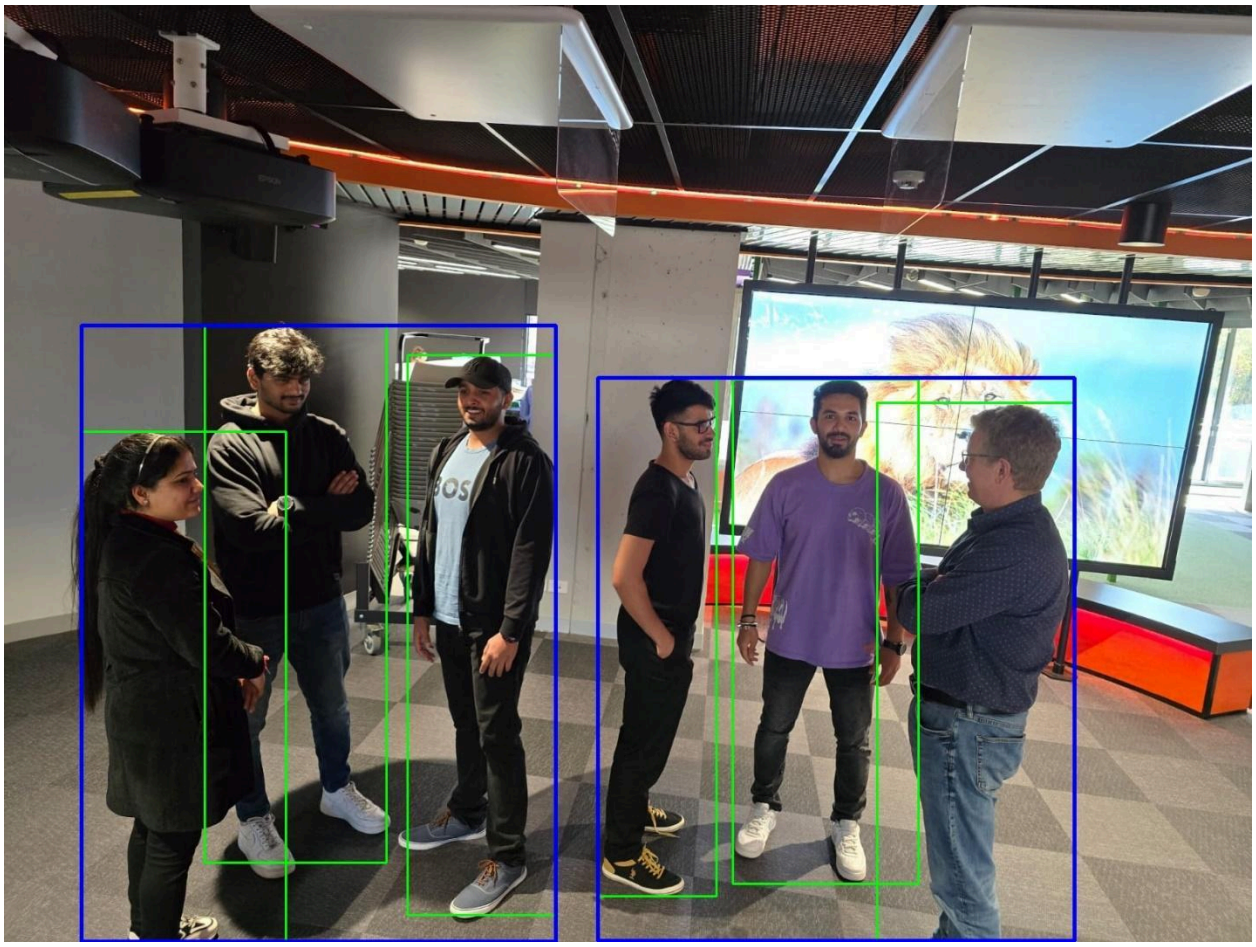

iStock
Credit: SolStock




iStock
530685719



Finally the group detection output on the final presentation day



Dashboard and Visualization

Visualization tools:
For this project, we focused on analyzing office space utilization by visualizing real-time occupancy data. We used Power BI to create a dashboard that shows how the office spaces are used, the busiest times, and what activities contribute most to the occupancy. This report will explain how we approached the project, we used different visualizations tools in our project.
Visualization Techniques Used:

1. **Scatter Plot**
   A scatter plot was used to track group locations and sizes in the office over time. This helped me see where larger groups gathered and when those areas were busiest.
2. **Line Chart**
   We used a line chart to monitor how the office space was used during different times of the day. This helped me understand peak hours and identify which times had less traffic.
3. **Heatmap**
   A heatmap was created to show the areas of the office with the highest and lowest occupancy. The color coding made it easy to quickly spot which spaces were most used and which were underutilized.
4. **Treemap**
   A treemap was used to show how different event types, like meetings or lunch breaks, affected office occupancy. Each block in the treemap represented a specific event and how many people participated.
5. **Gauge Chart**
   The gauge chart displayed the overall occupancy rate, helping me understand how full the office was at any given time. This was useful for checking if the office was nearing full capacity.
6. **Pie Chart**
   A pie chart was used to break down the occupancy by event type. It showed which activities, such as work sessions or snack breaks, were the most common in contributing to overall space usage.

Dataset Generation
Objective of Dataset Generation
The primary objective of generating a synthetic dataset was to compensate for the lack of access to real-time video data. The generated dataset needed to capture realistic occupancy patterns, including people count, room utilization, group dynamics, and event-based occupancy. This simulated data provides a basis for exploring key office space insights.

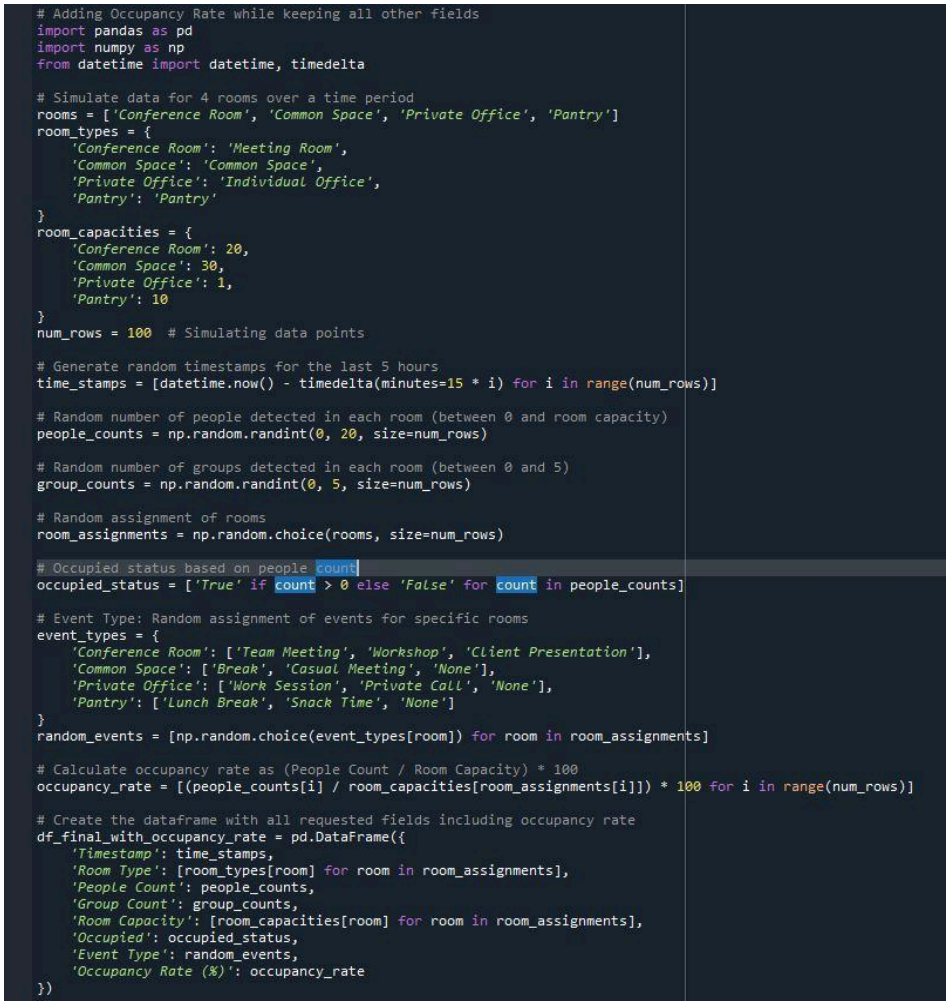
Python Code and Dataset Details
The dataset was created using a Python script that simulates data over a specified period. The script generates randomized data points across various fields to mimic real-life office usage. Below is a breakdown of each field:

- **Timestamp**: Reflects the specific time the data is recorded, allowing for trend analysis over time.

- **Room Type**: Rooms are categorized as Meeting Room, Common Space, Private Office, or Pantry, each with specific usage patterns and capacities.
- **Event Type**: Typical office activities such as Team Meeting, Lunch Break, and Workshop are assigned to room types, representing different occupancy scenarios.
- **People Count and Group Count**: Randomized within capacity limits to represent varied occupancy levels.
- **Occupied Status**: Boolean field indicating if a room is in use, based on the presence of people.
- **Room Capacity**: Sets a maximum occupancy for each room type (e.g., Meeting Room capacity is 20, Common Space is 10).
- **Occupancy Rate**: Calculated as (People Count / Room Capacity) * 100, giving a clear view of room utilization.

Code for Data Generation

The following Python code snippet shows the process of data generation:



```python
# Adding Occupancy Rate while keeping all other fields
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Simulate data for 4 rooms over a time period
rooms = ['Conference Room', 'Common Space', 'Private Office', 'Pantry']
room_types = {
    'Conference Room': 'Meeting Room',
    'Common Space': 'Common Space',
    'Private Office': 'Individual Office',
    'Pantry': 'Pantry'
}
room_capacities = {
    'Conference Room': 20,
    'Common Space': 30,
    'Private Office': 1,
    'Pantry': 10
}
num_rows = 100  # Simulating data points

# Generate random timestamps for the last 5 hours
time_stamps = [datetime.now() - timedelta(minutes=15 * i) for i in range(num_rows)]

# Random number of people detected in each room (between 0 and room capacity)
people_counts = np.random.randint(0, 20, size=num_rows)

# Random number of groups detected in each room (between 0 and 5)
group_counts = np.random.randint(0, 5, size=num_rows)

# Random assignment of rooms
room_assignments = np.random.choice(rooms, size=num_rows)

# Occupied status based on people count
occupied_status = ['True' if count > 0 else 'False' for count in people_counts]

# Event Type: Random assignment of events for specific rooms
event_types = {
    'Conference Room': ['Team Meeting', 'Workshop', 'Client Presentation'],
    'Common Space': ['Break', 'Casual Meeting', 'None'],
    'Private Office': ['Work Session', 'Private Call', 'None'],
    'Pantry': ['Lunch Break', 'Snack Time', 'None']
}
random_events = [np.random.choice(event_types[room]) for room in room_assignments]

# Calculate occupancy rate as (People Count / Room Capacity) * 100
occupancy_rate = [(people_counts[i] / room_capacities[room_assignments[i]]) * 100 for i in range(num_rows)]

# Create the dataframe with all requested fields including occupancy rate
df_final_with_occupancy_rate = pd.DataFrame({
    'Timestamp': time_stamps,
    'Room Type': [room_types[room] for room in room_assignments],
    'People Count': people_counts,
    'Group Count': group_counts,
    'Room Capacity': [room_capacities[room] for room in room_assignments],
    'Occupied': occupied_status,
    'Event Type': random_events,
    'Occupancy Rate (%)': occupancy_rate
})
```

Challenges and Solutions

- **Realism in Data**: Generating data that accurately reflects real office occupancy patterns posed a challenge. This was addressed by setting capacity limits and assigning events relevant to each room type.
- **Data Volume and Complexity**: Balancing the complexity of data with dashboard performance was crucial. Room types and events were limited to maintain a manageable dataset while ensuring diverse usage patterns.

Data Preparation for Power BI

Data Cleaning and Structuring

Before importing into Power BI, the dataset was cleaned and prepared. Below is an image of the cleaned dataset used in the dashboard.

	A	B	C	D	E	F	G	H	I
1	Timestamp	Hour	Room Type	People Count	Group Count	Room Capacity	Occupied	Event Type	Occupancy Rate (%)
2	3:14:10 PM	3:00:00 PM	Meeting Room	19	0	20	TRUE	Workshop	95
3	2:59:10 PM	2:00:00 PM	Pantry	2	1	10	TRUE	Snack Time	20
4	2:44:10 PM	2:00:00 PM	Meeting Room	7	3	20	TRUE	Workshop	35
5	2:29:10 PM	2:00:00 PM	Common Space	9	0	30	TRUE	Casual Meeting	30
6	2:14:10 PM	2:00:00 PM	Common Space	2	1	30	TRUE	Casual Meeting	6.666666667
7	1:59:10 PM	1:00:00 PM	Individual Office	0	0	2	FALSE	Private Call	0
8	1:44:10 PM	1:00:00 PM	Common Space	15	3	30	TRUE	Casual Meeting	50
9	1:29:10 PM	1:00:00 PM	Individual Office	1	0	2	TRUE	Work Session	50
10	1:14:10 PM	1:00:00 PM	Individual Office	0	0	2	FALSE	None	0
11	12:59:10 PM	12:00:00 PM	Common Space	14	0	30	TRUE	Casual Meeting	46.66666667
12	12:44:10 PM	12:00:00 PM	Meeting Room	10	0	20	TRUE	Workshop	50
13	12:29:10 PM	12:00:00 PM	Pantry	6	0	10	TRUE	Lunch Break	60
14	12:14:10 PM	12:00:00 PM	Individual Office	1	0	2	TRUE	None	50
15	11:59:10 AM	11:00:00 AM	Pantry	9	2	10	TRUE	Snack Time	90
16	11:44:10 AM	11:00:00 AM	Individual Office	0	0	2	TRUE	Work Session	0
17	11:29:10 AM	11:00:00 AM	Pantry	1	0	10	TRUE	None	10
18	11:14:10 AM	11:00:00 AM	Individual Office	2	1	2	TRUE	None	100
19	10:59:10 AM	10:00:00 AM	Pantry	8	3	10	TRUE	Lunch Break	80
20	10:44:10 AM	10:00:00 AM	Individual Office	0	1	2	FALSE	Private Call	0

- **Data Cleaning**: Outliers in people count or occupancy rate were removed, and timestamp formats were standardized.
- **Data Structuring**: Each row represented a unique data point with timestamp, room type, event, people count, and occupancy rate fields. This structure allowed Power BI to process the data for various visualization formats effectively.

Importing and Setting Up Data in Power BI

Once cleaned, the dataset was imported into Power BI. Relationships were set up where necessary, and data types were defined to ensure seamless functionality with Power BI's visuals.

Dashboard Development and Visualization

Dashboard Objectives

The Power BI dashboard was developed to meet the following objectives:

1. **Overview of Occupancy Levels**: Show the total number of people and groups present in the office.
2. **Space Utilization**: Highlight rooms with high or low utilization.
3. **Event-Based Analysis**: Allow users to see how events impact occupancy.

Dashboard Layout and Visuals

The dashboard includes various sections, each designed to offer unique insights:

Key Sections:

1. **Top KPI Metrics**:
    - **Total People**: Shows the overall number of individuals present.
    - **Average Group Size**: Reflects average group sizes across rooms.
    - **Total Groups**: Provides insights into group dynamics.
    - **Occupancy Rate Gauge**: Shows real-time workspace utilization.
2. **Room Type and Occupancy Analysis**:
    - A bar chart visualizes group count by room type, highlighting which rooms see the most activity.
3. **Event-Based Usage Patterns**:
    - A treemap shows occupancy rates by event type, while a pie chart provides a breakdown by event, helping users identify high-demand activities.

Interactive Filters

- **Room Type Filter**: Enables focus on specific rooms.
- **Event Type Filter**: Provides insights into how particular events impact occupancy.
- **Time Filter**: Allows users to focus on occupancy patterns for specific times.

Maintenance and Troubleshooting

Data Refresh

The dashboard is set to refresh every 15 minutes, pulling the latest data from the simulated dataset.

Performance Monitoring

- **Optimization**: Visuals are periodically reviewed to ensure they don't overload the system.
- **Troubleshooting**: Common issues like refresh failures or slow performance are monitored, with solutions like optimizing visuals or adjusting data parameters implemented as needed.

Troubleshooting Guide

- **Inaccurate Display**: Validate data calculations in Power BI to ensure correct occupancy and group metrics.

Future Recommendations

Data Source Integration

Once access to a real-time video feed becomes available, the dataset could be expanded to use actual occupancy data rather than simulated data. This would increase accuracy and allow for more detailed analysis.

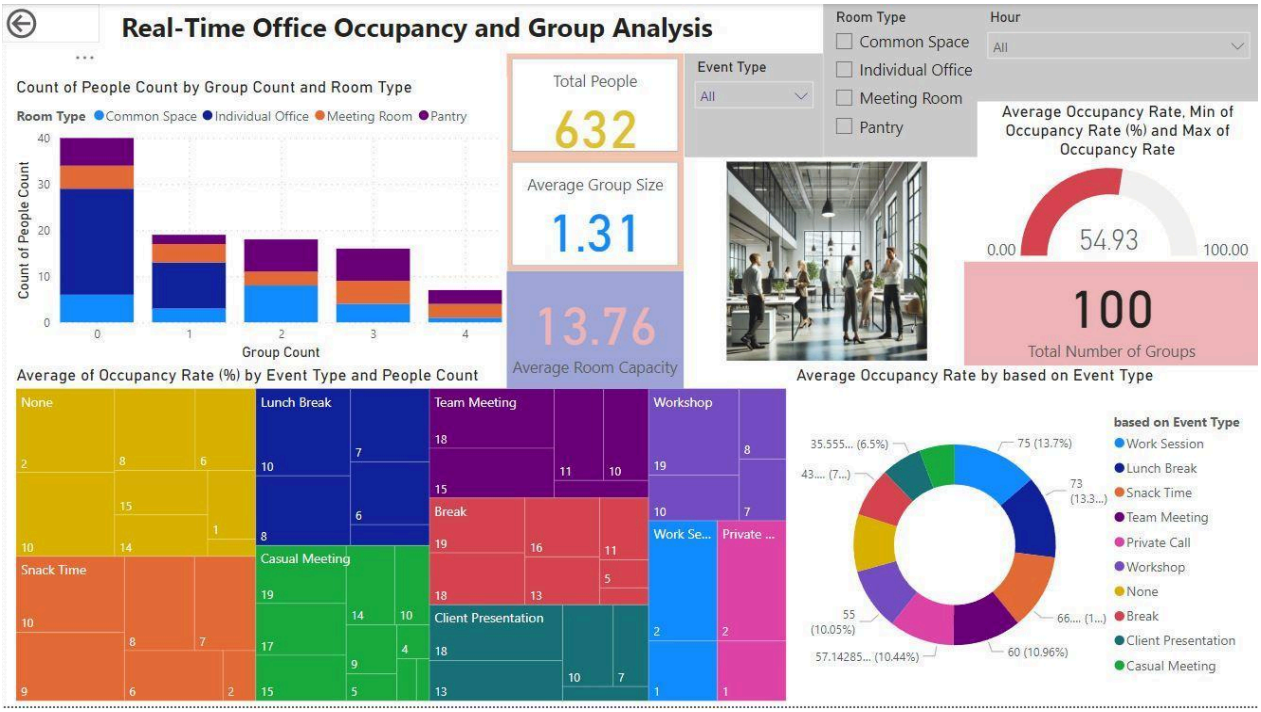
Advanced Visualizations

- **Time-Series Analysis**: Adding a time-series component could provide a view of historical trends.
- **Heatmaps**: Displaying heatmaps of occupancy could help identify popular areas within the office.

Machine Learning for Predictive Insights

Integrating machine learning could enable predictive insights, helping managers anticipate peak usage times and plan accordingly

Final Dashboard



Integration with Cisco overview

Purpose and Objectives

The primary aim of integrating Cisco Systems with Power BI is to harness the robust networking capabilities of Cisco alongside the advanced data analytics features of Power BI. This integration is designed to facilitate enhanced business intelligence that aids real-time decision-making and strategic planning.

Practical Implementations

1. **API Connection Setup:**
    - **Security Protocols:** To establish a secure and reliable connection between Cisco systems and Power BI, we implement OAuth 2.0 for authentication. This ensures that all data transfers are authenticated and authorized securely.
    - **API Endpoints Configuration:** We configure API endpoints on Cisco's servers to send data directly to Power BI. Each endpoint is tailored to match specific data streams, such as network performance metrics or security logs, ensuring that the data is correctly segmented and transmitted efficiently.
2. **Data Mapping:**
    - **Schema Alignment:** We meticulously map data schemas from Cisco's databases to Power BI data models. This involves identifying key data elements in Cisco's system that are valuable for analytics and mapping these to corresponding fields in Power BI. For instance, network traffic data from Cisco might be mapped to traffic analysis models in Power BI.
    - **Transformation Logic:** To ensure the data is analytics-ready upon arrival in Power BI, we implement transformation logic at the data collection stage. This includes converting timestamps, normalizing data formats, and aggregating metrics at the source to reduce processing load and improve responsiveness in Power BI dashboards.
3. **Data Flow Optimization:**
    - **Real-Time Streaming:** Leveraging Azure's real-time capabilities, we set up a streaming service that pushes data from Cisco to Power BI as it is generated. This minimizes latency and ensures dashboard updates are nearly instantaneous.
    - **Buffering Strategies:** To manage data bursts during high network traffic periods, we implement buffering strategies that temporarily store data before processing. This helps in managing load and ensures smooth data flow without loss.

Detailed Integration Steps of Cisco Systems with Power BI

Step-by-Step Practical Implementation

1. **API Connection and Security Setup:**
    - **Secure API Connectivity:** Establishing API connections between Cisco and Power BI is crucial. We use secure, encrypted channels to transmit data, ensuring that all

communications between Cisco devices and Power BI are protected against unauthorized access.
- **Authentication Mechanisms:** Implementing OAuth 2.0 protocol for authentication ensures that only authorized devices and services can access and transmit data. This setup helps in maintaining the integrity and confidentiality of the data exchanged.

2. **Data Flow Optimization:**

   - **Data Endpoint Configuration:** Configuring the API endpoints involves defining the data structure, ensuring compatibility between Cisco's output data format and Power BI's input requirements. This might include specifying JSON or XML formats, setting up data push intervals, and establishing error handling protocols.
   - **Real-Time Data Transmission:** Utilizing technologies like Azure Event Hubs or Kafka for real-time data streaming enables immediate data availability in Power BI. This setup is critical for operational dashboards that rely on up-to-the-minute data for network management and threat detection.
   - **Load Management:** Implementing throttling and backpressure mechanisms to manage data flow during peak load times prevents system overloads and ensures continuous data delivery without loss of service.

3. **Real-Time Data Integration:**

   - **Stream Analytics:** Setting up Azure Stream Analytics jobs helps in processing and analyzing streaming data from Cisco in real time. This allows for the implementation of complex event processing (CEP) logic on the stream, which can transform, enrich, and analyze the data before it hits the Power BI dashboards.
   - **Data Refresh Strategies:** Configuring data refresh settings in Power BI to handle streaming data effectively. This includes setting up appropriate refresh cycles that align with the pace of data change and business requirements, ensuring that dashboard users always have the latest information at their disposal.

4. **Testing and Validation:**

   - **Integration Testing:** Conducting comprehensive testing that includes scenario-based API testing, performance testing for data throughput, and user acceptance testing to ensure that the integration meets the functional and performance criteria.
   - **Data Validation:** Performing data validation involves checking the accuracy and timeliness of the data displayed on Power BI dashboards. This is critical to ensure that the business intelligence gathered is reliable and actionable.
   - **Feedback Loop Implementation:** Establishing a feedback mechanism to continuously monitor and adjust the integration based on user feedback and system performance data. This helps in iteratively improving the integration setup to better meet business needs.

Testing and Launch of Cisco Systems and Power BI Integration

Detailed Steps for Effective Testing and Implementation

1. **Testing and Validation:**
    - **Integration Testing:** Conduct thorough integration testing to ensure that the APIs between Cisco and Power BI are working as expected. This involves testing all endpoints for response times, data accuracy, and security. Tests simulate real-world scenarios to ensure the system performs under typical and peak loads.
    - **Performance Testing:** This testing phase focuses on the system's ability to handle large volumes of data efficiently and without degradation in performance. We use automated tools to generate and monitor traffic to the system, ensuring that it can manage expected and unexpected loads.
    - **Security Testing:** Security tests include penetration testing and vulnerability scanning to identify potential security flaws in the integration. This ensures that all data transmissions are secure and compliant with organizational security policies.
2. **Data Validation:**
    - **Accuracy Checks:** Validate the accuracy of data received in Power BI by comparing it with source data from Cisco systems. This step ensures that the data transformation and transmission processes do not introduce errors.
    - **Timeliness Checks:** Assess the timeliness of the data updates on Power BI dashboards to ensure that the real-time data integration meets business needs. This involves checking the system's ability to update dashboards promptly according to predefined intervals.
3. **Feedback and Adjustments:**
    - **Stakeholder Feedback:** Gather feedback from key stakeholders and users on the usability and functionality of the Power BI dashboards. This feedback is crucial for identifying areas of improvement and ensuring the system meets user expectations.
    - **Iterative Improvements:** Based on the feedback and test results, make necessary adjustments to the integration setup. This might involve tweaking API configurations, improving data handling procedures, or enhancing security measures.
4. **Final Setup and Launch:**
    - **Pre-Launch Review:** Conduct a final review of all integration settings, data flows, and user interfaces. Ensure everything is aligned with the project objectives and stakeholder expectations.
    - **Training and Documentation:** Provide training sessions for end-users and develop comprehensive documentation on the system’s operation, troubleshooting, and maintenance.
    - **Official Launch:** Roll out the integration in a controlled environment to monitor initial performance. Gradually scale up to full deployment as confidence in the system stability and performance increases.

Challenges and Solutions in Cisco Systems and Power BI Integration

In-depth Examination of Key Challenges and Strategic Solutions

1. **Challenge: Data Latency**
    - **Description:** One of the significant issues faced during the integration was data latency, where delays in data transmission affected the real-time capability of the Power BI dashboards. Timely data is critical for making informed decisions quickly in a dynamic business environment like Cisco's network operations.
    - **Practical Solution:**
        - **Optimized API Calls:** We refined the API interactions to enhance data fetching speeds. This involved optimizing query parameters to reduce the load and increase efficiency.
        - **Advanced Caching Mechanisms:** Implemented caching strategies where frequent data requests are temporarily stored to reduce the need to fetch data repeatedly from the source, thereby reducing load times and improving response rates.
2. **Challenge: Complex Integration Management**
    - **Description:** Managing the integration of two complex systems like Cisco and Power BI presented difficulties, especially in maintaining the integrity and security of data across different platforms.
    - **Practical Solution:**
        - **Robust Testing Protocols:** Adopted a rigorous testing regimen that included continuous integration/continuous deployment (CI/CD) practices to manage updates and integration changes smoothly.
        - **Comprehensive Documentation:** Developed detailed documentation of all integration processes, API specifications, and system configurations. This documentation serves as a reference to maintain consistency and educate new team members or stakeholders on the system's functionalities.
3. **Challenge: Security Compliance**
    - **Description:** Ensuring that the integration complies with both Cisco's and external regulatory security standards was crucial, given the sensitivity of the data being handled.
    - **Practical Solution:**
        - **Regular Security Audits:** Conducted periodic security audits to identify and address potential vulnerabilities within the integration. This proactive

approach ensures that the system remains secure against evolving threats.

- **Implementation of Advanced Security Features:** Enhanced security measures, including data encryption both at rest and in transit, multi-factor authentication for system access, and role-based access controls to ensure that only authorized personnel have access to sensitive data.

Special Notes

Special thanks to **Dr. Scott Mann** for lecturing and tutoring our team.