# Introduction

This Jupyter notebook demonstrates a comprehensive approach for generating synthetic data to simulate office occupancy over an entire year. The aim is to model real-world scenarios that can impact occupancy, such as:

* Meeting room bookings.
* Event activities.
* Staff attendance.

With considerations for external factors like Transportation availability and Weather conditions. The generated data will serve as a basis for training a predictive model using Facebook's Prophet, allowing us to forecast office occupancy with nuanced understanding and prediction capabilities. This document will guide you through the process of data generation, explaining the purpose and functionality of each code block, ensuring a clear understanding of how synthetic data can be utilized for occupancy prediction.

### Essential Python and Pandas Setup

Before we dive into generating our synthetic data, we start by importing the necessary Python libraries and modules. This setup includes:

* numpy and random: For numerical operations and generating random values to simulate variability in our data.
* datetime.time: To work with time objects, essential for scheduling meetings and events.
* pandas: Our primary library for data manipulation and analysis, enabling us to create and handle datasets effectively.
* Holiday and AbstractHolidayCalendar from pandas.tseries.holiday: These classes allow us to define custom holiday calendars based on specific rules, crucial for accurately modeling business days in Belgium.
* CustomBusinessDay from pandas.tseries.offsets: This tool helps us to create a custom business day offset, excluding weekends and defined holidays, ensuring our data generation aligns with actual office operation days.

By setting up these libraries, we lay the groundwork for creating a realistic synthetic dataset that mirrors the complexity of real-world office occupancy scenarios.

import numpy as np  
import random  
from datetime import time  
import pandas as pd  
from pandas.tseries.holiday import Holiday, AbstractHolidayCalendar  
from pandas.tseries.offsets import CustomBusinessDay

### Defining a Custom Holiday Calendar for Belgium

To accurately simulate office occupancy, it's crucial to account for public holidays when the office might be closed. This is accomplished by defining a custom holiday calendar for Belgium, which will be used to exclude these holidays from our synthetic data generation process.

#### How It Works:

* **BelgiumHolidayCalendar Class**: Inherits from AbstractHolidayCalendar, allowing us to specify the rules that define public holidays in Belgium. Each holiday is created using the Holiday class, where we specify the name of the holiday, and the month and day it occurs.
* **Holidays Included**: The calendar includes a range of Belgian public holidays, from "New Year's Day" on January 1st to "Christmas Day" on December 25th, among others. This comprehensive list ensures our dataset reflects the actual days the office would be closed.
* **Custom Business Day Offset**: With the belgium\_business\_day variable, we create a custom business day offset that excludes weekends and the defined Belgian holidays. This offset is pivotal in generating a realistic timeline for our synthetic data, focusing on actual working days.

By incorporating this custom holiday calendar in our data generation process, we ensure that the simulated office occupancy data reflects the typical operational schedule of an office in Belgium, enhancing the realism and accuracy of our predictive modeling efforts.

# Custom Holiday Calendar for Belgium  
class BelgiumHolidayCalendar(AbstractHolidayCalendar):  
 rules = [  
 Holiday("New Year's Day", month=1, day=1),  
 Holiday("Easter Monday", month=4, day=17),  
 Holiday("Labor Day", month=5, day=1),  
 Holiday("Ascension Day", month=5, day=25),  
 Holiday("Whit Monday", month=6, day=5),  
 Holiday("Belgium National Day", month=7, day=21),  
 Holiday("Assumption of Mary", month=8, day=15),  
 Holiday("All Saints' Day", month=11, day=1),  
 Holiday("Armistice Day", month=11, day=11),  
 Holiday("Christmas Day", month=12, day=25)  
 ]  
  
# Custom business day to exclude weekends and Belgian holidays  
belgium\_business\_day = CustomBusinessDay(calendar=BelgiumHolidayCalendar())

### Simulating External Factors: Transportation and Weather

To enhance the realism of our synthetic office occupancy data, we simulate external factors that can significantly impact staff attendance: transportation availability and weather conditions. Understanding these elements allows us to adjust our attendance figures to reflect real-world scenarios more accurately.

#### Transportation Availability

def generate\_transportation\_schedule(date):  
 # Simulate transportation availability  
 transportation\_factor = random.choice([0.8, 1, 1.2]) # Reduced, normal, or increased availability  
 return transportation\_factor

**Purpose**: Simulates the availability of transportation, which can affect how easily staff can commute to the office. Factors less than 1 represent reduced availability, while factors greater than 1 indicate increased availability.

### Weather Conditions

def generate\_weather\_condition(date):  
   
 # Define weather conditions for each season  
 spring\_conditions = ['Sunny', 'Rainy']  
 winter\_conditions = ['Sunny', 'Cloudy', 'Snow']  
 other\_conditions = ['Sunny', 'Cloudy']  
  
 # Extract the month from the given date  
 month = date.month  
  
 # Determine the season based on the month  
 if month in [3, 4, 5]: # Spring (March, April, May)  
 weather = random.choice(spring\_conditions)  
 elif month in [12, 1, 2]: # Winter (December, January, February)  
 weather = random.choice(winter\_conditions)  
 else: # Other months (June to November)  
 weather = random.choice(other\_conditions)  
  
 return weather

**Purpose**: Assigns weather conditions based on the season, influencing staff's decision to come to the office or participate in events.

### Adjusting Attendance for External Factors

def adjust\_attendance\_for\_factors(staff\_present, transportation\_factor, weather):  
   
 # Initialize the adjustment factor to 1 (no adjustment by default)  
 adjustment\_factor = 1.0  
  
 # Determine the adjustment factor based on weather and transportation  
 if weather == 'Rainy' and transportation\_factor < 1.0:  
 adjustment\_factor = 0.7 # Decrease attendance by 30%  
 elif weather == 'Rainy':  
 adjustment\_factor = 0.85 # Decrease attendance by 15%  
 elif weather == 'Sunny' and transportation\_factor > 1.0:  
 adjustment\_factor = 1.3 # Increase attendance by 30%  
 elif weather == 'Sunny':  
 adjustment\_factor = 1.15 # Increase attendance by 15%  
  
 # Calculate the adjusted staff and client counts  
 adjusted\_staff = int(staff\_present \* adjustment\_factor)  
  
 return adjusted\_staff

**Purpose**: Adjusts the expected staff presence based on the day's weather and transportation availability, providing a nuanced view of potential office occupancy.

## Daily Staff Attendance Simulation

def daily\_staff\_attendance(date, hour, transportation\_factor, weather, baseline\_staff=None):  
 avg\_staff = 30  
 std\_dev = 5   
  
 if baseline\_staff is None:  
 baseline\_staff = max(0, int(np.random.normal(avg\_staff, std\_dev)))  
  
 # Define a daily pattern for staff presence  
 daily\_pattern = {  
 8: -1, 9: -1, 10: 0, 11: 2, 12: -5, 13: 4, 14: 3, 15: 3, 16: 1, 17: 0  
 }  
 # Apply the daily pattern to hourly fluctuations  
 hourly\_fluctuation = daily\_pattern.get(hour.hour, 0)  
  
 staff\_present = max(0, baseline\_staff + hourly\_fluctuation)  
  
 adjusted\_staff = adjust\_attendance\_for\_factors(staff\_present, transportation\_factor, weather)  
 return [date, hour, adjusted\_staff]

**Purpose**: Generates hourly staff attendance data, incorporating fluctuations throughout the day and adjusting for external factors. This method simulates the dynamic nature of office occupancy, influenced by both internal schedules and external conditions.

These simulations collectively enable a detailed and realistic prediction model for office occupancy, taking into account not just the internal factors like meetings and events but also the external influences of transportation and weather.

## Simulating Meeting Room Bookings

For a comprehensive understanding of office occupancy, it's essential to consider the use of meeting rooms throughout the workday. This section of our simulation focuses on generating data for meeting room bookings, reflecting how meetings are scheduled and attended in a typical office environment.

#### Approach to Generating Meeting Room Bookings:

def daily\_meeting\_room\_bookings(date):  
 # Adjust probability based on the day of the week (e.g., more meetings on Mondays)  
 day\_of\_week = date.weekday()  
 base\_probability = 0.8 if day\_of\_week == 0 else 0.4  
  
 # Generate a random number of bookings using a Poisson distribution  
 num\_bookings = np.random.poisson(lam=base\_probability \* 5)  
  
 # Create a list of booking records  
 bookings = []  
 for \_ in range(num\_bookings):  
 # Generate a random time between 8:00 AM and 4:00 PM  
 h = random.randint(8, 16)  
 meeting\_time = time(hour=h, minute=0, second=0)  
   
   
 # Choose a random meeting duration (30, 60 or 180 minutes) and number of attendees (2 to 20)  
 duration = random.choice([30, 60, 180])  
 attendees = random.randint(2, 20)  
   
 # Add the booking record to the list  
 bookings.append([date, meeting\_time, duration, attendees])  
  
 return bookings

* **Probability Adjustment**: The likelihood of a meeting being scheduled varies with the day of the week, with Mondays typically seeing a higher number of meetings. This is modeled by adjusting the base probability of bookings.
* **Meeting Details**: For each meeting, the start time, duration, and number of attendees are randomly determined. This introduces a realistic variance in meeting characteristics, from short check-ins to longer strategy sessions, and small team gatherings to larger departmental meetings.
* **Data Output**: The function returns a list of booking records for a given day, with each record detailing the date, start time, duration, and attendees of a meeting.

By incorporating this simulation into our dataset, we can better understand the dynamics of office space usage and more accurately predict overall occupancy levels. This data not only aids in forecasting but also in planning resources and managing office space effectively.

## Simulating Event Activities within the Office

Event activities, ranging from team-building exercises to training sessions, play a crucial role in determining daily office occupancy. This segment focuses on simulating these activities, taking into account seasonal variations to mirror the increased frequency of events during certain times of the year.

#### Generating Data for Event Activities:

def daily\_event\_activities(date):  
 # Adjust probability for seasonal variation (more events in summer)  
 month = date.month  
 base\_probability = 0.4 if month in [6, 7, 8] else 0.1  
  
 if random.random() < base\_probability:  
 # Generate a random start time during working hours (8:00 AM to 6:00 PM)  
 start\_hour = random.randint(8, 17) # 17 to ensure the event starts before 6:00 PM  
 start\_time = time(hour=start\_hour, minute=0, second=0)  
  
 event\_type = random.choice(['Team Building', 'Client Meeting', 'Training Session', 'Celebration'])  
 expected\_attendance = random.randint(5, 50)  
  
 return [[date, start\_time, event\_type, expected\_attendance]]  
   
 return []

* **Seasonal Variation**: Recognizes that events are more likely during the summer months, adjusting probabilities to reflect this seasonal trend.
* **Event Details**: For each event, the function randomly selects a start time, type, and expected attendance, ensuring a variety of events are represented in the synthetic dataset.
* **Output**: Returns a list of event records for the specified date, with each record detailing the event's date, start time, type, and expected attendance. If no event is scheduled for a given day, an empty list is returned.

Incorporating event activities into our occupancy simulation allows for a more dynamic and realistic representation of how office spaces are utilized. This data is instrumental in forecasting occupancy levels, facilitating effective space management, and enhancing the overall workplace environment.

## Generating Yearly Synthetic Data for Office Occupancy

To create a comprehensive dataset that captures the nuances of office occupancy throughout an entire year, we employ a function that generates synthetic data on a day-to-day basis. This function is versatile, capable of simulating staff attendance, meeting room bookings, and event activities, depending on the type of data generator passed to it.

### Yearly Data Generation Process:

work\_hours = [time(hour=i, minute=0, second=0) for i in range(8, 18)]  
  
def generate\_yearly\_data(start\_date, end\_date, data\_generator):  
 data = []  
 current\_date = start\_date  
 baseline\_staff = None # Initialize baseline staff for each day  
  
 while current\_date <= end\_date:  
 # Check if the current date is a weekend or a holiday  
 is\_weekend\_or\_holiday = current\_date.weekday() >= 5 or current\_date in belgium\_business\_day.holidays  
   
 # Skip weekends and holidays  
 if not is\_weekend\_or\_holiday:  
 if data\_generator == daily\_staff\_attendance:  
 # Generate transportation and weather factors once per day  
 transportation\_factor = generate\_transportation\_schedule(current\_date)  
 weather = generate\_weather\_condition(current\_date)  
  
 for hour in work\_hours:  
 # Generate staff attendance data with adjustments for transportation and weather  
  
 staff\_data = data\_generator(current\_date, hour, transportation\_factor, weather, baseline\_staff)  
 data.append(staff\_data)  
 else:  
 # Generate data for meeting room bookings or evening activities  
  
 daily\_data = data\_generator(current\_date)  
 data.extend(daily\_data)  
  
 current\_date += pd.Timedelta(days=1)  
  
 return pd.DataFrame(data)

* **Flexible Data Generation**: The function adapts to generate different types of data based on the data\_generator argument, allowing for the simulation of various aspects of office occupancy.
* **Daily Operations**: It iteratively generates data for each business day within the specified date range, skipping weekends and holidays to mirror real-world office activity.
* **Adjustments for External Factors**: Specifically for staff attendance, it calculates daily adjustments based on transportation availability and weather conditions, providing a realistic depiction of factors influencing office occupancy.
* **Output**: The function returns a pandas DataFrame containing the generated data, which can be further analyzed or used as input for predictive modeling. This approach to synthetic data generation offers a detailed and dynamic representation of office occupancy, essential for accurate forecasting and efficient office management strategies.

## Compiling and Exporting the Synthetic Office Occupancy Data

After generating synthetic data for meeting room bookings, event activities, and staff attendance, we compile these datasets to cover an entire year, from January 1, 2023, to December 31, 2023. This comprehensive dataset provides a granular view of daily office occupancy, vital for occupancy prediction and space management.

### Steps to Compile the Yearly Data:

1. **Data Generation**: Utilizing the previously defined functions, we generate data for each category over the specified date range, ensuring a detailed and realistic simulation of office activities throughout the year.
2. **Data Structuring**: For each dataset, we define appropriate column names to clearly represent the data. This includes details such as the date, start times, durations, and attendance figures, making the data intuitive and easy to analyze.
3. **Data Exportation**: Each dataset is exported as a CSV file, allowing for easy storage, sharing, and further analysis. This step ensures the data is accessible for occupancy forecasting, planning, and decision-making processes.

# Generate data for an entire year  
start\_date = pd.Timestamp('2023-01-01')  
end\_date = pd.Timestamp('2023-12-31')  
  
# Generate datasets  
meeting\_room\_bookings\_year = generate\_yearly\_data(start\_date, end\_date, daily\_meeting\_room\_bookings)  
event\_activities\_year = generate\_yearly\_data(start\_date, end\_date, daily\_event\_activities)  
staff\_attendance\_year = generate\_yearly\_data(start\_date, end\_date, daily\_staff\_attendance)  
  
# Define columns for the generated data  
  
meeting\_room\_bookings\_year.columns = ['Date', 'Start Time', 'Duration (min)', 'Room Capacity' ]  
  
event\_activities\_year.columns = ['Date', 'Start Time', 'Event Type', 'Expected Attendance']  
staff\_attendance\_year.columns = ['Date', 'Report time', 'Bodies Present'] #, 'Transportation', 'Weather'   
  
meeting\_room\_bookings\_year.to\_csv('./data/meeting\_room\_bookings\_year.csv', index=False)  
event\_activities\_year.to\_csv('./data/event\_activities\_year.csv', index=False)  
staff\_attendance\_year.to\_csv('./data/staff\_attendance\_year.csv', index=False)

### Exported Datasets:

* **Meeting Room Bookings**: Contains records of all meeting room bookings, detailing the date, start time, duration, and room capacity.
* **Event Activities**: Lists all event activities, providing the date, start time, event type, and expected attendance.
* **Staff Attendance**: Shows daily staff attendance records, including the date, report time, and the number of bodies present.

### Example Data Exportation Code:

# Display sample data  
print("Sample Meeting Room Bookings:\n", meeting\_room\_bookings\_year.head())  
print("\nSample Evening Activities:\n", event\_activities\_year.head())  
print("\nSample Staff Attendance:\n", staff\_attendance\_year.head())

Sample Meeting Room Bookings:  
 Date Start Time Duration (min) Room Capacity  
0 2023-01-03 11:00:00 180 11  
1 2023-01-04 09:00:00 60 6  
2 2023-01-05 12:00:00 60 14  
3 2023-01-06 09:00:00 60 3  
4 2023-01-06 12:00:00 180 3  
  
Sample Evening Activities:  
 Date Start Time Event Type Expected Attendance  
0 2023-01-13 11:00:00 Team Building 11  
1 2023-01-16 12:00:00 Training Session 41  
2 2023-02-01 11:00:00 Training Session 33  
3 2023-02-13 16:00:00 Training Session 36  
4 2023-03-21 15:00:00 Celebration 49  
  
Sample Staff Attendance:  
 Date Report time Bodies Present  
0 2023-01-02 08:00:00 39  
1 2023-01-02 09:00:00 31  
2 2023-01-02 10:00:00 34  
3 2023-01-02 11:00:00 29  
4 2023-01-02 12:00:00 20

* **Sample Meeting Room Bookings**: [Displays the first few records of the meeting room bookings dataset]
* **Sample Evening Activities**: [Displays the first few records of the event activities dataset]
* **Sample Staff Attendance**: [Displays the first few records of the staff attendance dataset]

This process not only highlights the depth and breadth of the synthetic data generated but also underscores the potential applications of such data in predicting office occupancy. By analyzing these datasets, organizations can gain valuable insights into occupancy patterns, enabling more informed decision-making regarding space utilization and office management.

## Conclusion

In this Jupyter notebook, we embarked on a comprehensive journey to generate synthetic data that simulates office occupancy, factoring in various real-world conditions such as meeting room bookings, event activities, and staff attendance. By incorporating external influences like transportation availability and weather conditions, we've created a dataset that closely mirrors the dynamic nature of office occupancy.

### Key Takeaways:

* **Customization and Realism**: Through the customization of holiday calendars, simulation of transportation and weather conditions, and the generation of detailed meeting and event data, we've laid the groundwork for highly realistic occupancy forecasting.
* **Versatility of Data**: The generated data spans a full year, providing a rich dataset for training predictive models, such as Facebook's Prophet, to forecast office occupancy with a high degree of accuracy and reliability.
* **Insights for Office Management**: The insights derived from analyzing this synthetic data can inform office space planning, resource allocation, and the overall management of office environments, leading to more efficient and effective utilization of space.

### Future Directions:

* **Predictive Modeling**: With the datasets in hand, the next step involves applying predictive modeling techniques to forecast future occupancy levels, enabling proactive office management.
* **Data Enrichment**: Further enriching the dataset with additional variables or integrating real-world data could enhance the model's accuracy and applicability to various scenarios.
* **Decision-Making Support**: The ultimate goal is to utilize the insights gained from occupancy predictions to support decision-making processes, from daily operations to long-term strategic planning.

In conclusion, the process outlined in this notebook demonstrates the power of synthetic data in understanding and predicting office occupancy. By carefully simulating the complexities of office life, we pave the way for data-driven decisions that can significantly improve office management and employee satisfaction. As we move forward, the potential applications of this data in predictive analytics and AI will undoubtedly open new avenues for optimizing office environments in an ever-evolving workplace landscape.