Documentation: AI Model

By JeeTwo

Preface

In this report we will be talking about the analysis we have prepared for the project of the “Office occupancy tracker”. We will start off by talking about the AI we will be using, after this we will talk about the web application and its corresponding API. And finally, we will talk about hosting the solution and the backup solutions for the project.

Table of contents

Contents

[Final Report: Analysis of occupancy tracker 1](#_Toc159343048)

[Preface 2](#_Toc159343049)

[Table of contents 3](#_Toc159343050)

[1. Cost analysis 4](#_Toc159343051)

[1.1. Web application 4](#_Toc159343052)

1. Data Generation

This report 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.

* 1. 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

* 1. 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())

* 1. 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.

* 1. 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.

* 1. 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.

* 1. 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.

* 1. 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.

* 1. 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.

* 1. 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.

* 1. 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)

* 1. 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.
  1. Example Data Exportation Code:

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.

* 1. Conclusion

In this section, 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.

1. Introduction to Data Cleaning for Office Occupancy Prediction

In this Jupyter notebook, we embark on the essential process of data cleaning and preprocessing with the goal of preparing datasets for accurate office occupancy prediction. Our focus is on three key datasets: meeting room bookings, event activities, and staff attendance, collected over the span of a year. The objective of this phase is to refine and enhance the quality of our data by addressing anomalies, ensuring data consistency, and implementing necessary transformations. This meticulous preparation is crucial for the subsequent analysis and modeling stages, particularly for employing predictive modeling techniques such as Facebook's Prophet. Through this notebook, we will navigate the steps of importing and preprocessing data, merging datasets for a unified analysis framework, transforming and engineering features for improved model performance, and conducting thorough data visualization and outlier detection. Our ultimate aim is to ensure the data is primed for generating accurate and actionable occupancy forecasts, thereby facilitating more informed decision-making in office space management and optimization.

* 1. Setting Up the Environment for Data Cleaning

Before diving into the data cleaning process, it's crucial to set up our working environment by importing the necessary libraries and tools. This setup is foundational for handling, analyzing, and visualizing the data effectively throughout the notebook. Here's a brief overview of the libraries we're using and their roles in our data cleaning journey:

* **Pandas**: The cornerstone of our data manipulation and analysis, allowing us to read, preprocess, and aggregate our datasets.
* **NumPy**: Provides support for numerical operations, including mathematical transformations needed during data preprocessing.
* **Scikit-learn's OneHotEncoder**: Useful for encoding categorical variables as part of feature engineering, enhancing model interpretability.
* **Matplotlib & Seaborn**: Our go-to libraries for data visualization, enabling us to plot distributions, trends, and correlations within the data, making insights more accessible.
* **SciPy's Stats Module**: Offers statistical functions, crucial for identifying and handling outliers within our datasets.
* **Pandas' Holiday and CustomBusinessDay**: These classes help us manage and account for holidays and custom business days, ensuring our data reflects actual office occupancy patterns.

This comprehensive suite of tools equips us to tackle the various aspects of data cleaning and preparation, setting the stage for a detailed analysis and modeling process.

import pandas as pd  
import numpy as np  
from sklearn.preprocessing import OneHotEncoder  
import matplotlib.pyplot as plt  
import seaborn as sns  
from pandas.tseries.holiday import Holiday, AbstractHolidayCalendar  
from pandas.tseries.offsets import CustomBusinessDay  
from scipy import stats  
import matplotlib.pyplot as plt

* + 1. Loading the Datasets

With our environment set up, the next step in our data cleaning journey is to load the datasets into our notebook. These datasets form the core of our analysis and modeling work, each capturing different facets of office occupancy:

* **Meeting Room Bookings**: This dataset includes details on the bookings of meeting rooms within the office over the year, such as the date and time of bookings, duration, and room capacity.
* **Event Activities**: Records various events occurring throughout the year, detailing the type of event, its expected attendance, and timing.
* **Staff Attendance**: Tracks the daily attendance of staff, providing insights into the overall occupancy trends within the office.

# Read datasets  
meeting\_room\_bookings\_year = pd.read\_csv('./data/meeting\_room\_bookings\_year.csv')  
event\_activities\_year = pd.read\_csv('./data/event\_activities\_year.csv')  
staff\_attendance\_year = pd.read\_csv('./data/staff\_attendance\_year.csv')

* + 1. Preprocessing Staff Attendance Data

The initial phase of data cleaning involves preprocessing the staff attendance dataset. This step is vital for transforming the data into a more analyzable format, which will significantly aid in our subsequent analysis and predictive modeling efforts. Here's what we accomplish in this preprocessing step:

* **Combining Date and Time into a Single Datetime Column**

To capture the precise moments of staff attendance, we combine the Date and Report time columns into a single DateTime column. This consolidation is crucial for time series analysis, allowing us to observe attendance patterns down to the hour.

* **Aggregating Attendance Data**

With the DateTime column established, we proceed to aggregate the data to sum up the total number of bodies present at each recorded datetime. This aggregation provides us with a clearer view of occupancy levels throughout the year.

# Preprocess Staff Attendance Data  
staff\_attendance\_year['DateTime'] = pd.to\_datetime(staff\_attendance\_year['Date'].astype(str) + ' ' + staff\_attendance\_year['Report time'].astype(str))  
staff\_agg = staff\_attendance\_year.groupby('DateTime').agg({'Bodies Present': 'sum'}).reset\_index()   
staff\_agg.head()

DateTime 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

* + 1. Processing Meeting Room Bookings Data

After addressing the staff attendance dataset, our next focus shifts to refining the meeting room bookings data. This dataset encapsulates information on the usage of meeting spaces within the office, a key component in understanding overall occupancy dynamics. The processing steps applied here aim to render the data more suitable for time series analysis and predictive modeling.

* **Creating a Unified DateTime Column**

Similar to the staff attendance data, we consolidate the Date and Start Time into a single DateTime column for the meeting room bookings. This step ensures that each booking's timing is precisely captured, facilitating a detailed analysis of meeting room occupancy over time.

* **Summarizing Room Capacity** To get a comprehensive view of the capacity utilized during meetings, we aggregate the data by the newly created DateTime column, summing up the Room Capacity for each datetime. This aggregation highlights the total meeting room capacity engaged at any given moment, offering insights into the demand for meeting spaces.

# Process Meeting Room Bookings Data  
meeting\_room\_bookings\_year['DateTime'] = pd.to\_datetime(meeting\_room\_bookings\_year['Date'].astype(str) + ' ' + meeting\_room\_bookings\_year['Start Time'].astype(str))  
room\_capacity\_sum = meeting\_room\_bookings\_year.groupby('DateTime')['Room Capacity'].sum().reset\_index()

* + 1. Processing Evening Activities Data

The processing of the evening activities data follows a similar methodology to the meeting room bookings, focusing on events that contribute to the overall occupancy of the office space. These activities, ranging from team-building exercises to training sessions, significantly influence the dynamic use of office areas. Here's how we process this dataset:

* **Creating a Unified DateTime Column**

For the event activities data, we also combine the Date and Start Time into a DateTime column. This amalgamation is essential for aligning event data with the temporal framework used for staff attendance and meeting room bookings, ensuring consistency across all datasets.

* **Summarizing Expected Attendance**

Understanding the impact of events on office occupancy requires a grasp of how many individuals are expected at these activities. Therefore, we aggregate the event data by DateTime, summing up the Expected Attendance for each event. This gives us a clear picture of the anticipated occupancy due to events at any given time.

# Process Evening Activities Data  
event\_activities\_year['DateTime'] = pd.to\_datetime(event\_activities\_year['Date'].astype(str) + ' ' + event\_activities\_year['Start Time'].astype(str))  
expected\_attendance\_sum = event\_activities\_year.groupby('DateTime')['Expected Attendance'].sum().reset\_index()

* 1. Preparing Data for Occupancy Prediction with Prophet

After processing our datasets, the next crucial step is to prepare the data for occupancy prediction using Facebook's Prophet. This involves merging the staff attendance, meeting room bookings, and event activities data into a single dataframe, transforming the data where necessary, and selecting relevant features for the prediction model. Here's how we accomplish this:

* + 1. Merging Datasets

The datasets are merged based on the DateTime column, now standardized as ds across all datasets, to create a comprehensive view of occupancy-related activities. This unified dataframe, df\_prophet, serves as the foundation for our predictive modeling.

# Merge Data for Prophet  
df\_prophet = staff\_agg.rename(columns={'DateTime': 'ds', 'Bodies Present': 'y\_original'})  
room\_capacity\_sum = room\_capacity\_sum.rename(columns={'DateTime': 'ds', 'Room Capacity': 'total\_room\_capacity'})  
expected\_attendance\_sum = expected\_attendance\_sum.rename(columns={'DateTime': 'ds', 'Expected Attendance': 'total\_expected\_attendance'})  
  
df\_prophet = df\_prophet.merge(room\_capacity\_sum, how='left', on='ds').fillna({'total\_room\_capacity': 0})  
df\_prophet = df\_prophet.merge(expected\_attendance\_sum, how='left', on='ds').fillna({'total\_expected\_attendance': 0})

### Calculating Expected Occupancy

To get a holistic measure of expected office occupancy, we sum the total\_room\_capacity and total\_expected\_attendance, creating an expected\_occupancy feature. This reflects the combined impact of meetings and events on occupancy.

# Calculate Expected Occupancy for that time  
df\_prophet['expected\_occupancy'] = df\_prophet['total\_room\_capacity'] + df\_prophet['total\_expected\_attendance']

* + 1. Transforming the Target Variable

A logarithmic transformation is applied to the target variable (y\_original) to normalize its distribution, enhancing the model's performance. This transformed variable is denoted as y.

# Apply a logarithmic transformation to the target variable  
df\_prophet['y'] = np.log1p(df\_prophet['y\_original'])

* + 1. Feature Engineering

To enrich our dataset for the Prophet model, we extract the month and day\_of\_week from the ds column. These features can serve as additional regressors, potentially improving the model's accuracy by accounting for seasonal and weekly occupancy patterns.

# Add Month and DayOfWeek to later use as regressors for prophet  
df\_prophet['month'] = pd.to\_datetime(df\_prophet['ds']).dt.month  
  
df\_prophet['day\_of\_week'] = pd.to\_datetime(df\_prophet['ds']).dt.dayofweek

#### Creating a Workday Indicator

df\_prophet['is\_workday'] = df\_prophet['day\_of\_week'].apply(lambda x: 0 if x >= 5 else 1)

**Purpose**: To enhance the forecasting model by introducing a binary indicator (is\_workday) that differentiates weekdays (workdays) from weekends (non-workdays), based on the assumption that Saturday (5) and Sunday (6) are typically non-working days.

* + 1. Final Dataset Selection

We finalize the dataframe by selecting columns relevant to our prediction model: ds (datetime), y (log-transformed occupancy), y\_original (actual occupancy), expected\_occupancy, month, and day\_of\_week.

# Select Relevant Columns  
df\_prophet = df\_prophet[['ds', 'y', 'y\_original' , 'expected\_occupancy', 'month', 'day\_of\_week', 'is\_workday']]   
df\_prophet.head(5)

ds y y\_original expected\_occupancy month \  
0 2023-01-02 08:00:00 3.688879 39 0.0 1   
1 2023-01-02 09:00:00 3.465736 31 0.0 1   
2 2023-01-02 10:00:00 3.555348 34 0.0 1   
3 2023-01-02 11:00:00 3.401197 29 0.0 1   
4 2023-01-02 12:00:00 3.044522 20 0.0 1   
  
 day\_of\_week is\_workday   
0 0 1   
1 0 1   
2 0 1   
3 0 1   
4 0 1

* + 1. Visualizing the Target Variable Distribution

Understanding the distribution of our target variable is a crucial step in data preprocessing. We focus on visualizing the number of bodies present, which serves as our occupancy measure.

# Distribution of the Target Variable (original and log-transformed)  
plt.figure(figsize=(12, 6))  
plt.subplot(1, 2, 1)  
sns.histplot(df\_prophet['y\_original'], kde=True)  
plt.title('Distribution of Original Target Variable (y\_original)')  
  
plt.subplot(1, 2, 2)  
sns.histplot(df\_prophet['y'], kde=True)  
plt.title('Distribution of Log-Transformed Target Variable (y log-transformed)')  
plt.show()

A graph of a function

Description automatically generated with medium confidence

* + 1. Original Target Variable Distribution

The first histogram illustrates the distribution of the occupancy count before any transformations. This visualization helps identify skewness and assess whether the raw data follow a normal distribution, which is a common assumption for many statistical models.

* + 1. Log-Transformed Target Variable Distribution

The second histogram displays the data after applying a logarithmic transformation. This transformation often helps normalize the data, reducing skewness and making patterns more evident and consistent, which can be particularly beneficial for linear models and algorithms that assume normality.

The side-by-side comparison of these histograms is instrumental in determining the necessity and effectiveness of the log transformation, guiding the choice of appropriate data preprocessing steps for predictive modeling.

* + 1. Analyzing the Target Variable Over Time

Examining the target variable through time enables us to capture trends, patterns, and anomalies in office occupancy. The plot illustrates how the log-transformed count of bodies present fluctuates on a daily basis throughout the year.

# Target Variable Over Time  
plt.figure(figsize=(10, 5))  
plt.plot(df\_prophet['ds'], df\_prophet['y'])  
plt.title('Target Variable Over Time')  
plt.xlabel('Date')  
plt.ylabel('Bodies Present')  
plt.show()

A graph showing a graph of a graph

Description automatically generated with medium confidence

* + 1. Trends and Seasonality in Office Occupancy

The visualization potentially reveals underlying trends and recurring patterns which could be indicative of seasonality or specific cycles in office attendance. Understanding these patterns is essential for accurate forecasting and can inform decisions on resource management and space planning.

* + 1. Occupancy Variability

The degree of variability in the data also becomes evident through this time series plot. Peaks and troughs may correspond to specific events or changes in office policy, suggesting areas for deeper investigation.

Through such visual analysis, we gain valuable insights into the temporal dynamics of office occupancy, which is pivotal in crafting a robust predictive model.

The output confirms whether there are any missing datetimes that would need to be addressed before proceeding with further analysis or modeling. The absence of missing datetimes suggests our dataset is comprehensive and ready for the next stages of our data preparation process.

* + 1. Examining Trends of the Target Variable Over Time

The graph provides a comparative view of the office occupancy trends over time, displaying both the original and log-transformed target variable values. This visual comparison is key for several reasons:

#### Understanding Data Fluctuations

The original data's trend line allows us to observe the raw fluctuations in occupancy, offering an unadjusted view of the data's variance and potential outliers.

#### Impact of Log Transformation

Overlaying the log-transformed data, we can see how this transformation affects the scale and visibility of trends within the data. The transformed line typically appears smoother, which may be beneficial for identifying long-term trends and seasonality.

#### Preparing for Predictive Modeling

By comparing these two trend lines, we gain insights into how the data transformation might influence the predictive modeling process. This is especially relevant for models that assume or prefer normally distributed data.

# Trends Over Time  
plt.figure(figsize=(15, 6))  
plt.plot(df\_prophet['ds'], df\_prophet['y\_original'], label='Original')  
plt.plot(df\_prophet['ds'], df\_prophet['y'], label='Log-Transformed', alpha=0.7)  
plt.xlabel('Date')  
plt.ylabel('Values')  
plt.title('Trend of Target Variable Over Time')  
plt.legend()  
plt.show()

A blue and orange line graph

Description automatically generated

* + 1. Analyzing Occupancy by Day of the Week

The bar chart illustrates the average occupancy within the office across different days of the week. Such an analysis is crucial for several reasons:

#### Weekly Occupancy Patterns

It highlights the patterns of attendance and office use throughout a standard workweek, which can inform resource allocation and office space planning.

#### Day-to-Day Variations

Understanding the variations in average occupancy from day to day helps in identifying peak occupancy days, which could influence scheduling decisions for meetings and events.

# Average occupancy by day of the week  
sns.barplot(x='day\_of\_week', y='y\_original', data=df\_prophet)  
plt.title('Average Occupancy by Day of the Week')  
plt.xlabel('Day of the Week')  
plt.ylabel('Average Occupancy')  
plt.show()

A graph of different colored bars

Description automatically generated

* + 1. Exploring Monthly Trends in Office Occupancy

The bar chart provides a monthly breakdown of average office occupancy, offering valuable insights into how occupancy levels fluctuate throughout the year.

#### Seasonal Impact on Occupancy

The visualization can reveal seasonal trends, such as higher or lower occupancy during certain months, which may correlate with business cycles, holidays, or vacation periods.

#### Planning and Resource Allocation

By understanding which months have higher occupancy, office managers can better plan for resource allocation, space utilization, and even energy management within the office.

#### Informing Forecasting Models

These insights are also beneficial for forecasting models like Prophet, as they underscore the importance of considering monthly effects when predicting future occupancy levels.

Such visual analyses are instrumental in uncovering the underlying seasonal patterns in office occupancy, aiding in strategic planning and predictive modeling efforts.

# Average occupancy by month  
sns.barplot(x='month', y='y\_original', data=df\_prophet)  
plt.title('Average Occupancy by Month')  
plt.xlabel('Month')  
plt.ylabel('Average Occupancy')  
plt.show()

A graph of different colored bars

Description automatically generated

* + 1. Identifying and Visualizing Outliers in Office Occupancy Data

Detecting outliers is a critical step in data cleaning, as these anomalies can significantly affect the performance of predictive models. The graph illustrates our approach to identifying outliers in the office occupancy data using z-scores, a statistical measure that indicates how many standard deviations an element is from the mean.

#### Outlier Detection Using Z-Scores

Outliers are defined as data points that are several standard deviations away from the mean. By calculating z-scores, we can objectively identify these outliers. Typically, a z-score above 3 or below -3 is considered to indicate an outlier.

#### Visualization of Outliers

The plot displays the office occupancy data over time, with outliers marked in red. This visual representation helps us quickly identify any unusual spikes or drops in occupancy that deviate significantly from the general pattern.

#### Implications of Outliers

Removing or appropriately handling these outliers is crucial, as they could represent data entry errors, special one-off events, or other irregularities that do not reflect normal occupancy patterns. By addressing these outliers, we can ensure a more accurate and robust predictive model for future occupancy levels.

# Calculate z-scores  
z\_scores = stats.zscore(df\_prophet['y'])  
  
# Get absolute values to identify outliers on both tails  
abs\_z\_scores = np.abs(z\_scores)  
  
# Identify outliers  
outliers = df\_prophet[abs\_z\_scores > 3]  
  
  
# Plotting to visualize the outliers  
plt.figure(figsize=(10, 5))  
plt.plot(df\_prophet['ds'], df\_prophet['y'], label='Data')  
plt.scatter(outliers['ds'], outliers['y'], color='red', label='Outliers')  
plt.title('Outliers in the Data')  
plt.xlabel('Date')  
plt.ylabel('Bodies Present')  
plt.legend()  
plt.show()

A graph of data with blue lines and red dots

Description automatically generated

### Correlation Analysis for Predictive Modeling

In predictive analytics, understanding the relationships between different variables is crucial. Correlation analysis helps us determine how various features may influence the target variable, which in this case is the office occupancy level.

#### Interpreting the Correlation Matrix

A correlation matrix provides a visual and numerical representation of the correlation coefficients between pairs of variables. These coefficients range from -1 to 1, where:

* **1** indicates a perfect positive correlation.
* **-1** indicates a perfect negative correlation.
* **0** indicates no correlation.

#### Visualizing Correlations with a Heatmap

We use a heatmap to visualize the correlation matrix, making it easier to identify strong or significant correlations between the variables. This can inform feature selection for our predictive model, indicating which variables may be worth including as predictors.

# Correlation Analysis  
df\_prophet.head()  
correlation\_matrix = df\_prophet.corr()  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')  
plt.title('Correlation Matrix')  
plt.show()

A screenshot of a graph

Description automatically generated

#### Insights for Model Development

This correlation analysis is a vital step in preparing our data for occupancy prediction with Prophet. By identifying which variables are most strongly associated with occupancy levels, we can tailor our model to focus on the most influential factors, potentially improving its accuracy and effectiveness.

The heatmap displayed above shows the correlation matrix for the variables within our dataset. It highlights the relationships that will be considered when developing the predictive model to forecast future office occupancy trends.

### Finalizing the Data Cleaning Process

After conducting a thorough analysis and visualization of the data, the final step in our data cleaning notebook involves removing outliers and saving the cleaned dataset. This ensures that our predictive model is trained on high-quality data, which is essential for generating accurate forecasts.

#### Outlier Removal

Outliers can have a disproportionate effect on the model's training process and can lead to overfitting or poor generalization on unseen data. By filtering out these extreme values using our calculated z-scores, we retain only the data points that are within three standard deviations from the mean, which typically encompasses around 99.7% of data in a normal distribution.

# Remove outliers from the dataframe  
df\_prophet = df\_prophet[abs\_z\_scores <= 3]

#### Saving the Cleaned Dataset

Once the outliers are removed, we save the resulting cleaned dataframe to a CSV file. This file will serve as the input for our occupancy prediction models, providing a solid and reliable foundation for our forecasts.

# Save the merged dataframe  
df\_prophet.to\_csv('./data/df\_prophet.csv')

By diligently cleaning and preparing the data, we increase the likelihood of developing a robust and reliable predictive model. The saved dataset is now ready for the next stage in our project, where we will apply Facebook's Prophet to forecast office occupancy levels based on the patterns and relationships we've uncovered in our analysis.

* 1. Conclusion: Data Integrity for Predictive Modeling

The data cleaning process we've completed is a testament to the importance of meticulous preparation in predictive analytics. We've imported our raw datasets, conducted preprocessing to create a unified temporal framework, transformed and engineered features for enhanced model compatibility, and visualized the data to understand underlying patterns and detect outliers.

Our thorough analysis included visualizing trends, analyzing the distribution of the target variable, and examining the relationships between different features through correlation analysis. This comprehensive approach not only informed our understanding of the data but also highlighted the importance of each preprocessing step in the context of occupancy prediction.

By removing outliers from the dataset, we've ensured that the data feeding into our predictive model reflects true occupancy patterns, free from the distortion of extreme anomalies. The saved, cleaned dataset is primed for the next phase, where predictive modeling techniques, such as Facebook's Prophet, can be applied to forecast future occupancy trends with greater accuracy.

The cleaned dataset is not just an input for a model; it represents a refined lens through which we can project and plan for future occupancy needs. As we move forward to the modeling stage, we do so with the confidence that comes from a foundation of quality, integrity, and detailed understanding of our data.

1. Forecasting Office Occupancy with Prophet

In this report, we utilize Facebook's Prophet, a powerful and flexible forecasting tool, to predict office occupancy based on our cleaned and processed dataset. Prophet is designed to handle the intricacies of time series data with daily and seasonal trends, making it an ideal choice for predicting complex patterns such as those found in office attendance and room bookings.

The process begins with loading the preprocessed dataset, ensuring datetime formats are correct, and then splitting the data into training and testing sets. We configure the Prophet model with appropriate seasonality and additional regressors to capture the nuances of our occupancy data. Once the model is trained, we make predictions on the test set and visualize the results, comparing the forecasted occupancy against the actual data.

To provide a comprehensive evaluation of the model's performance, we compute several accuracy metrics, including MAE (Mean Absolute Error), RMSE (Root Mean Square Error), MAPE (Mean Absolute Percentage Error), and the R-squared statistic. These metrics will help us understand the model's precision and predictive power.

Finally, we save the trained model for future forecasting, enabling us to make occupancy predictions with the click of a button. This notebook encapsulates the end-to-end process of building a time series forecasting model, from data preparation to model evaluation and deployment.

* 1. Initializing the Forecasting Process

As we set out to predict office occupancy, our first step is to gather the tools that will power our analysis. We import a suite of Python libraries and modules, each playing a pivotal role in the forecasting workflow:

* **Prophet**: Developed by Facebook, this library is tailored for forecasting time series data with its robust handling of seasonality, trends, and holidays.
* **Pandas**: Our data manipulation workhorse, it allows us to structure, explore, and manipulate the data easily.
* **Matplotlib**: This plotting library will enable us to visualize the data and the forecasting results, providing clear insights into the performance of our model.
* **NumPy**: Essential for numerical operations, especially when transforming or reverting transformations on our data.
* **Scikit-learn Metrics**: These functions allow us to quantitatively assess the performance of our model through various error metrics.
* **Pickle**: A module for saving our model, which ensures that we can quickly load the trained model for future predictions without retraining.

Equipped with these tools, we're ready to begin transforming our preprocessed data into forecasts that can inform decision-making and strategy for office management.

from prophet import Prophet  
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error  
import pickle

* 1. Loading the Preprocessed Dataset for Forecasting

With our environment prepared, we now load the preprocessed dataset that will be used for forecasting office occupancy. This dataset, saved from the previous stages of cleaning and preparation, contains the time series data that Prophet will use to fit its model.

By inspecting the first few rows with df\_prophet.head(), we ensure that the data is loaded correctly and get a quick glimpse of its structure. This verification step is critical to confirm that the data aligns with our expectations and is ready for the modeling phase.

# Load the dataset  
df\_prophet = pd.read\_csv('./data/df\_prophet.csv')  
df\_prophet.head()

Unnamed: 0 ds y y\_original expected\_occupancy \  
0 0 2023-01-02 08:00:00 3.688879 39 0.0   
1 1 2023-01-02 09:00:00 3.465736 31 0.0   
2 2 2023-01-02 10:00:00 3.555348 34 0.0   
3 3 2023-01-02 11:00:00 3.401197 29 0.0   
4 4 2023-01-02 12:00:00 3.044522 20 0.0   
  
 month day\_of\_week is\_workday   
0 1 0 1   
1 1 0 1   
2 1 0 1   
3 1 0 1   
4 1 0 1

* 1. Preparing the Time Series Data for Prophet

To ensure compatibility with the Prophet forecasting model, we must confirm that the datetime column, ds, is in the correct format. Prophet requires this column to be a datetime type without any timezone information—referred to as "timezone-naive."

This step involves converting the ds column to datetime format and removing any timezone information that may be attached, thereby meeting Prophet's data specifications. This process is vital for the model to correctly interpret the timestamps and for the successful fitting and forecasting of time series data.

# Ensure 'ds' is datetime and timezone-naive  
df\_prophet['ds'] = pd.to\_datetime(df\_prophet['ds']).dt.tz\_localize(None)

### Splitting the Data for Training and Testing

Before we can train our forecasting model, we need to divide our dataset into two subsets: one for training the model and another for evaluating its performance. This split allows us to understand how well the model can predict new, unseen data.

We allocate 80% of the data for training, which provides the model with a substantial historical context to learn the occupancy patterns. The remaining 20% is reserved for testing, giving us a clear measure of the model's forecasting accuracy.

By carefully splitting the data, we ensure that our model's performance evaluation is both rigorous and fair, providing a trustworthy assessment of its predictive capabilities.

# Split the data into training and testing sets  
train\_ratio = 0.8  
split\_index = int(len(df\_prophet) \* train\_ratio)  
train = df\_prophet.iloc[:split\_index]  
test = df\_prophet.iloc[split\_index:]

* 1. Configuring the Prophet Model

We start by setting up the Prophet model with the following configurations to ensure it can effectively model the daily and hourly seasonality observed in office occupancy data:

* **Daily Seasonality Enabled**: This setting allows the model to account for variations in occupancy that occur on a day-to-day basis, which is crucial for capturing the regular fluctuations associated with standard working hours and off-hours.
* **Changepoint Prior Scale**: We set this parameter to 0.01 to control the model's flexibility in detecting changes in the trend. A smaller value makes the model less sensitive to changes, helping to prevent overfitting to minor fluctuations and focusing on more significant shifts in occupancy trends.
* **Custom Hourly Seasonality**: To further refine the model's accuracy, we add a custom hourly seasonality component. This is particularly important for office occupancy forecasting, where there can be distinct patterns within the working day, such as peak occupancy times. We set the period to 1/24 to represent the hourly cycle within a day and choose a Fourier order of 8. This Fourier order allows the model to capture the primary occupancy patterns without becoming overly complex.

These settings are designed to optimize the Prophet model for the specific characteristics of office occupancy data, ensuring that both broad trends and finer-grained daily and hourly patterns are accurately captured.

# Initialize the Prophet model with all settings  
model = Prophet(daily\_seasonality=True, changepoint\_prior\_scale=0.02)  
model.add\_seasonality(name='hourly', period=2, fourier\_order=50)

<prophet.forecaster.Prophet at 0x25ccb8d0b90>

* 1. Enhancing the Prophet Model with Additional Regressors

To further refine our office occupancy forecasts, we incorporate additional regressors into the Prophet model. These regressors provide the model with more context, allowing it to account for factors that significantly influence occupancy beyond the basic time series trends and seasonality.

* **Adding Additional Regressors**: We enhance the model by introducing several additional regressors: is\_workday, expected\_occupancy, day\_of\_week, and month. Each of these factors is expected to have a distinct impact on office occupancy, from daily and monthly patterns to variations based on whether a day is a workday.

# Add additional regressors  
additional\_regressors = [ 'is\_workday', 'expected\_occupancy', 'day\_of\_week', 'month']   
for regressor in additional\_regressors:  
 model.add\_regressor(regressor)

* **Training the Model**: With the additional regressors configured, we proceed to fit the model to our training dataset. This step allows the model to learn the underlying patterns in the data, including how the additional regressors influence occupancy.

# Fit the model to the training data  
model.fit(train)

15:59:27 - cmdstanpy - INFO - Chain [1] start processing  
15:59:27 - cmdstanpy - INFO - Chain [1] done processing

<prophet.forecaster.Prophet at 0x25ccb8d0b90>

* **Forecasting Future Occupancy**: After training, we prepare the testing dataset by excluding the target variable (y) to simulate a real forecasting scenario. We then use the trained model to predict future occupancy for the dates within our testing set.

# Make predictions on the testing set  
future = test.drop(columns=['y'])   
forecast = model.predict(future)

* 1. Insights:
* By integrating additional regressors, we provide the model with a richer set of data points to consider, enabling more accurate and nuanced forecasts.
* The process of fitting the model and making predictions mirrors real-world forecasting tasks, where models are trained on historical data and used to predict future outcomes based on similar conditions.
* The inclusion of is\_workday as a regressor is particularly noteworthy, as it allows the model to differentiate between workdays and weekends or holidays, which is crucial for accurate occupancy forecasting in an office context. ```

This approach exemplifies how to leverage the Prophet model's flexibility to incorporate external factors into time series forecasting, aiming for enhanced prediction accuracy and relevance to specific use cases like office occupancy.

### 

* 1. Visual Comparison of Actual vs. Forecasted Occupancy

The visualization of actual versus forecasted office occupancy provides a clear picture of the model's performance. By plotting both the actual values and the forecasted values on a logarithmic scale, we gain insights into the accuracy of our predictions.

#### Code Explanation:

* **Visualization of Forecasting Results**: The graph illustrates the comparison between the actual occupancy data and the values predicted by the model. The use of a logarithmic scale helps in comparing growth rates and percentage changes more effectively.
* **Confidence Intervals**: The shaded area represents the confidence intervals (yhat\_lower and yhat\_upper) of the forecasts, providing a visual sense of the model's certainty in its predictions.
* **Chart Details**: The plot includes labels for the axes (Date and Values), a title ('Actual vs Forecast (Logarithmic scale)'), and a legend to distinguish between the actual and forecasted data.

# Compare forecast to actual values  
plt.figure(figsize=(10, 6))  
plt.plot(test['ds'], test['y'], label='Actual')  
plt.plot(forecast['ds'], forecast['yhat'], label='Forecast')  
plt.fill\_between(forecast['ds'], forecast['yhat\_lower'], forecast['yhat\_upper'], alpha=0.3)  
plt.xlabel('Date')  
plt.ylabel('Values')  
plt.title('Actual vs Forecast (Logarithmic scale)')  
plt.legend()  
plt.show()

A graph with blue and orange lines

Description automatically generated

#### Insights:

* **Model Evaluation**: This visual comparison is an integral part of model evaluation, as it allows us to quickly assess how well the forecasted trends and patterns align with the actual occupancy data.
* **Forecasting Accuracy**: Areas where the forecast diverges significantly from the actual data can identify periods where the model may need refinement or where external factors may have influenced occupancy levels.
* **Decision Making**: For office management, such visualizations can be instrumental in making informed decisions based on the forecasted occupancy, helping to plan for future space utilization and resource allocation.
  1. Visualizing Forecast Accuracy on the Original Scale

After training our forecasting model and making predictions, it's essential to assess the model's performance by comparing the forecasted values with the actual data.

### Reverting Log Transformations for Comparison

We initially applied a logarithmic transformation to stabilize the variance in our occupancy data. For a meaningful comparison between the predicted and actual values, we need to revert this transformation to the original scale:

# Revert the log transformation on the forecasted values  
forecast['yhat\_original'] = np.expm1(forecast['yhat'])  
forecast['yhat\_lower\_original'] = np.expm1(forecast['yhat\_lower'])  
forecast['yhat\_upper\_original'] = np.expm1(forecast['yhat\_upper'])  
  
# Plot the actual vs forecasted values  
plt.figure(figsize=(10, 6))  
plt.plot(test['ds'], np.expm1(test['y']), label='Actual') # Ensure to revert the test['y'] as well since it's on log scale  
plt.plot(forecast['ds'], forecast['yhat\_original'], label='Forecast')  
plt.fill\_between(forecast['ds'], forecast['yhat\_lower\_original'], forecast['yhat\_upper\_original'], alpha=0.3)  
plt.xlabel('Date')  
plt.ylabel('Values')  
plt.title('Actual vs Forecast (Original Scale)')  
plt.legend()  
plt.show()

A graph showing a blue and orange line

Description automatically generated

* 1. Evaluating Model Performance with Error Metrics

After transforming our predicted and actual occupancy values back to their original scale, we proceed to quantitatively assess the forecasting model's accuracy. This is done using two common statistical error metrics: Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

#### Calculating Error Metrics

To quantify the forecast accuracy, we calculate:

* **Mean Absolute Error (MAE)**: This metric provides the average magnitude of errors in a set of forecasts, without considering their direction (positive or negative). It is a measure of accuracy for continuous variables and is particularly useful when we want to understand the error size in the same units as the data itself.
* **Root Mean Square Error (RMSE)**: This metric provides a measure of the average magnitude of the error, giving a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

By reporting both MAE and RMSE, we gain a more comprehensive understanding of the model's performance, capturing both the average error and the variability of the errors.

# y\_true are the actual values and y\_pred are the predicted values  
  
y\_true = np.expm1(test['y']).values  
y\_pred = np.expm1(forecast['yhat']).values  
  
mae = mean\_absolute\_error(y\_true, y\_pred)  
rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))  
  
  
print(f"MAE: {mae}")  
print(f"RMSE: {rmse}")

MAE: 3.814677783318604  
RMSE: 4.948092363386668

#### Interpreting the Output

The given output indicates the average error (MAE) and the standard deviation of the errors (RMSE) between the forecasted and actual occupancy numbers:

The output of the error metrics is as follows:

* **MAE**: A value of 3.814677783318604 suggests that, on average, the model's predictions are approximately 3.81 occupancy counts off from the actual counts.
* **RMSE**: A value of 4.948092363386668 indicates the typical deviation of the forecast errors. The higher value compared to MAE suggests there are some larger errors in the dataset.

These metrics serve as benchmarks for the model's current predictive performance. Depending on the context and the acceptable error thresholds for occupancy predictions, these values can inform whether the model's accuracy is sufficient or if further improvements are necessary.

Understanding the magnitude of these errors in the context of your specific application is essential. If the errors are within an acceptable range, the model can be considered reliable for practical use. If not, these metrics can guide further model refinement or prompt a review of the underlying data and model assumptions.

### Saving the Trained Prophet Model

After training and validating our Prophet model, we want to save it for future use. This way, we don't need to retrain the model every time we wish to make predictions, which saves time and computational resources.

with open('./model/prophet\_model.pkl', 'wb') as file:  
 pickle.dump(model, file)

The result is a saved model file, prophet\_model.pkl, which contains all the information needed to load the trained model at a later date and make predictions without having to retrain from scratch.

### Conclusion: Ready for Future Predictions

With our Prophet model trained, evaluated, and saved, we conclude this phase of our forecasting project. We have successfully navigated through data preprocessing, model tuning, prediction, and evaluation, culminating in the serialization of our model for future use.

This serialized model stands as a milestone, encapsulating our analytical efforts and serving as a ready-to-use tool for predicting office occupancy. It can be quickly loaded to provide insights, support decision-making, or integrate with operational systems to manage office space efficiently.

As we wrap up this notebook, we look forward to leveraging the model to its full potential, confident in the robust foundation we have built through careful data science practices.