Machine Learning Model for OccupancyRates and Demand in the Hospitality Industry

Project Documentation

Submitted by

Ishita Jindal (21BCB0061)

ishita.jindal2021@vitstudent.ac.in

Anjali Srivastava (21BDS0069)

anjali.srivastava2021@vitstudent.ac.in

INDEX

ABSTRACT

The hospitality industry has evolved significantly since the inception of the first hotels in the late 18th century. From the humble beginnings of the City Hotel in New York City in 1794 to the opulent Tremont Hotel in Boston in 1829, the hotel industry has continually transformed. This evolution has been marked by changes in interior design, building structures, and amenities, culminating in the establishment of modern hotels. Today, these establishments are integral to the tourism and transportation sectors, contributing to various related activities. In a parallel narrative, the contemporary hospitality landscape witnesses a groundbreaking innovation in the form of an environmental-based solution for hotel occupancy prediction. This innovative system combines diverse environmental variables such as temperature, humidity, light, CO2 levels, and humidity ratio with occupancy status to forecast hotel occupancy rates and demand accurately. It introduces regression models to enable precise demand forecasting, optimizing resource allocation. Sustainability is at the core of this approach, aligning with environmental goals, and continuous monitoring and interpretability ensure ongoing accuracy. Ultimately, this forward-thinking method enhances guest experiences and contributes to sustainability within the hospitality industry, bridging the gap between historical hotels and cutting-edge occupancy and demand prediction techniques.

INTRODUCTION

In the realm of hotel occupancy forecasting, two primary categories of approaches have traditionally held sway: historical booking models and advanced booking models. Historical booking models tackle the forecasting challenge by treating it as a time series modeling problem, while advanced booking models rely on reservations data and the concept of "Pick-Up." This concept refers to the incremental increase in bookings, denoted as N, from the present to a future day, T, for which there are already K reservations. Consequently, the occupation forecast for day T is simply K + N.

Historically, a plethora of statistical techniques have been employed in this context. Methods like ARIMA and Holt-Winters exponential smoothing have been utilized for monthly hotel occupancy forecasts, consistently delivering low Mean Squared Error results. Some innovative techniques combine long-term forecasts via Holt-Winters with short-term forecasts derived from observations, leading to ensemble predictions. Neural Networks have also found application in time series prediction, excelling in scenarios such as forecasting Japanese citizens' travel to Hong Kong.

Notably, the Pick-Up method has been implemented in forecasting sold rooms for 7, 14, 30, and 60 days of reserves, often incorporating day-of-the-week considerations. Furthermore, a Monte-Carlo model has been proposed for occupancy forecasting at an Egyptian hotel, highlighting the versatility of statistical techniques in predicting occupancy rates and demand.

While statistical techniques have proven effective, their application often demands substantial statistical expertise and time-consuming procedures, including the analysis of correlograms. Additionally, some methods rely on expensive commercial software. In light of these challenges, this paper advocates the utilization of machine learning algorithms to construct predictive

models for occupancy rates and demand in the hospitality industry. These models are designed to be accessible to hotel personnel without requiring advanced statistical training. Moreover, they can be packaged into cost-effective applications or executed on cloud platforms, offering the hospitality sector an affordable and efficient means of forecasting.

The remainder of this paper is organized as follows: it commences with a brief introduction to the algorithms employed, followed by an overview of the dataset's structure and characteristics. Subsequently, the experimental results are presented and discussed, leading to our concluding remarks.

SOFTWARE REQUIREMENTS

Programming Language : Python,HTML,CSS,FLASK

Operating System : Windows

IDE editor : Jupyter Notebook

RAM required : 4GB or more

INTRODUCTION TO TECHNOLOGIES USED

Python

Python stands out as an exemplary choice for working with Machine Learning and AI models due to its abundance of built-in libraries that can be readily utilized with minimal need for extensive implementation and coding. Python, in essence, is a high-level, interpreted, interactive, and object-oriented scripting language. Notably, Python excels in readability, favoring English words over punctuation and featuring fewer syntactic complexities than many other programming languages.

Python operates in an interpreted manner, where code is processed at runtime by the interpreter, obviating the need for pre-compilation, a trait akin to PERL and PHP. Furthermore, Python offers an interactive environment, enabling direct interaction with the interpreter for program development. This language champions an object-oriented approach, encapsulating code within objects, and it serves as an excellent choice for beginner-level programmers, facilitating the creation of diverse applications, from basic text processing to web browsers and games.

Key features of Python encompass its ease of learning, boasting a limited number of keywords, straightforward structure, and a well-defined syntax that accelerates the learning curve. Python code is designed to be easily readable and comprehensible, enhancing code visibility. Maintenance of Python source code is relatively hassle-free, and it features an extensive standard library that is highly portable and compatible across various platforms, including UNIX, Windows, and Macintosh.

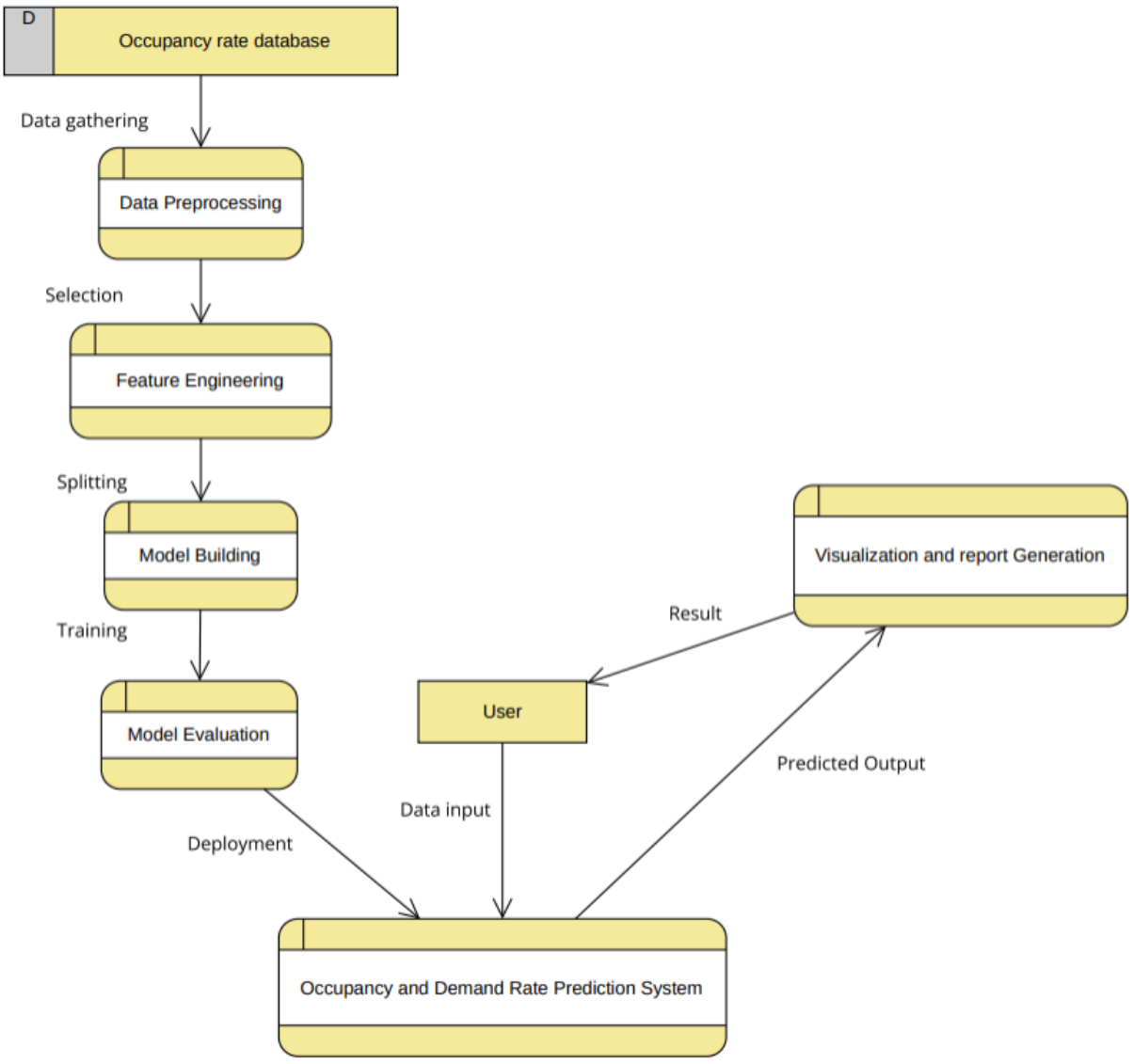
Python provides an interactive mode, facilitating the testing and debugging of code snippets interactively. It exhibits portability, as it can run on a diverse range of hardware platforms while maintaining a consistent interface. Python's extendability allows for the incorporation of low-level modules into the interpreter, enabling programmers to enhance and customize their tools for

increased efficiency.

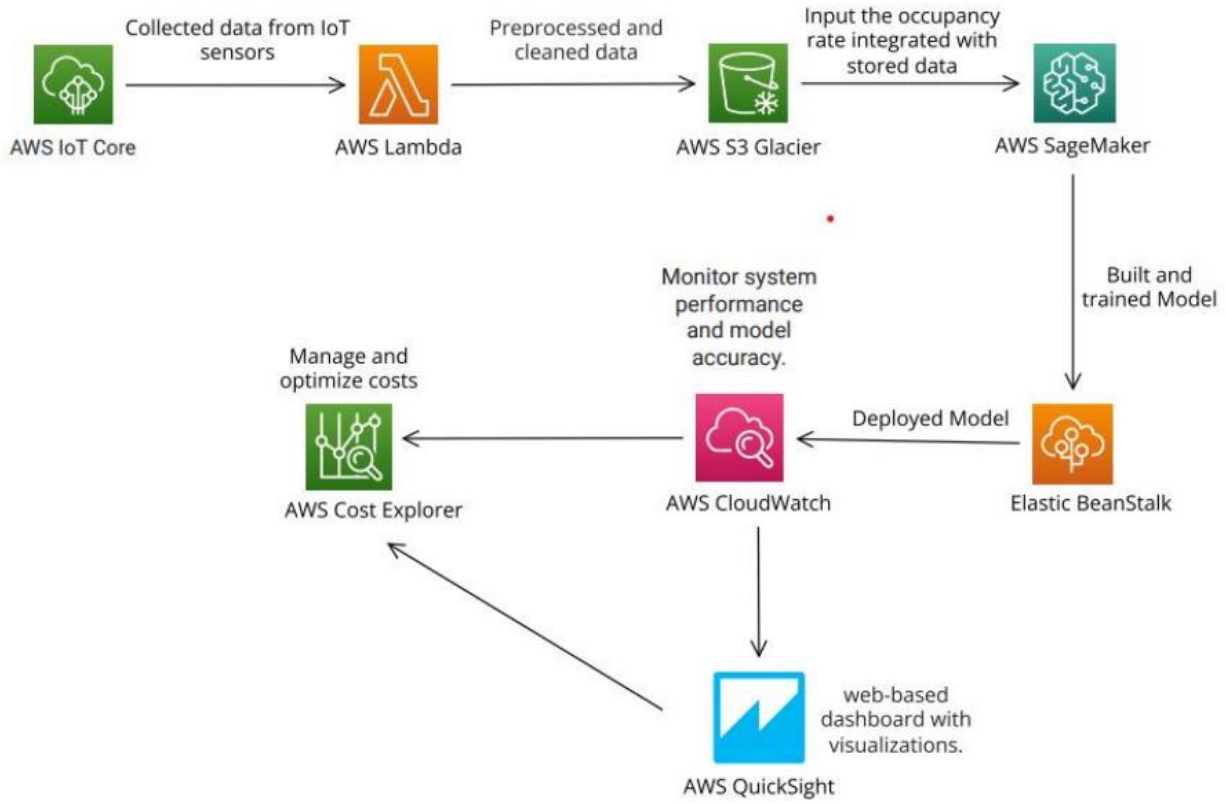
Moreover, Python offers support for interfacing with major commercial databases and the creation of graphical user interface (GUI) applications, which can be easily adapted and deployed on various operating systems and window systems. Python's scalability is notable, as it offers a structured and supportive environment for managing large programs, distinguishing itself from the limitations of shell scripting.

FLOW CHART

A flowchart is simply a graphical representation of steps. It shows steps in sequential order and is widely used in presenting the flow of algorithms, workflow or processes. Typically, a flowchart shows the steps as boxes of various kinds, and their order by connecting them with arrows.



SOLUTION ARCHITECTURE DIAGRAM



MODEL BUILDING

Activity 1: Importing the Model Building Libraries

Importing all the important libraries needed for this project:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
```

Activity 2: Loading the model

```
datatrain=pd.read_csv('datatraining.txt')
datatest1=pd.read_csv('datatest.txt')
datatest2=pd.read_csv('datatest2.txt')
```

Activity 3: Head and tail

```
datatrain.head()
```

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
1	2015-02-04 17:51:00	23.18	27.2720	426.0	721.25	0.004793	1
2	2015-02-04 17:51:59	23.15	27.2675	429.5	714.00	0.004783	1
3	2015-02-04 17:53:00	23.15	27.2450	426.0	713.50	0.004779	1
4	2015-02-04 17:54:00	23.15	27.2000	426.0	708.25	0.004772	1
5	2015-02-04 17:55:00	23.10	27.2000	426.0	704.50	0.004757	1

```
datatrain.tail()
```

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
8139	2015-02-10 09:29:00	21.05	36.0975	433.0	787.250000	0.005579	1
8140	2015-02-10 09:29:59	21.05	35.9950	433.0	789.500000	0.005563	1
8141	2015-02-10 09:30:59	21.10	36.0950	433.0	798.500000	0.005596	1
8142	2015-02-10 09:32:00	21.10	36.2600	433.0	820.333333	0.005621	1
8143	2015-02-10 09:33:00	21.10	36.2000	447.0	821.000000	0.005612	1

Activity 4: null values and info

```
datatrain.isnull().sum()
```

```
date            0
Temperature     0
Humidity        0
Light           0
CO2             0
HumidityRatio   0
Occupancy       0
dtype: int64
```

```
datatrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8143 entries, 1 to 8143
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           8143 non-null   object 
 1   Temperature    8143 non-null   float64
 2   Humidity       8143 non-null   float64
 3   Light          8143 non-null   float64
 4   CO2            8143 non-null   float64
 5   HumidityRatio  8143 non-null   float64
 6   Occupancy      8143 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 508.9+ KB
```

Activity 5: Data shape and describe

```
datatrain.shape
```

```
(8143, 7)
```

```
datatrain.describe()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	8143.000000	8143.000000	8143.000000	8143.000000	8143.000000	8143.000000
mean	20.619084	25.731507	119.519375	606.546243	0.003863	0.212330
std	1.016916	5.531211	194.755805	314.320877	0.000852	0.408982
min	19.000000	16.745000	0.000000	412.750000	0.002674	0.000000
25%	19.700000	20.200000	0.000000	439.000000	0.003078	0.000000
50%	20.390000	26.222500	0.000000	453.500000	0.003801	0.000000
75%	21.390000	30.533333	256.375000	638.833333	0.004352	0.000000
max	23.180000	39.117500	1546.333333	2028.500000	0.006476	1.000000

Activity 6: split the date and check the head and drop the date column

```
[ ] datatrain[["Year","Month","Day"]]=datatrain["date"].str.split("-",expand=True)
```

```
[ ] datatrain.head()
```

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy	Year	Month	Day
1	2015-02-04 17:51:00	23.18	27.2720	426.0	721.25	0.004793	1	2015	02	04 17:51:00
2	2015-02-04 17:51:59	23.15	27.2675	429.5	714.00	0.004783	1	2015	02	04 17:51:59
3	2015-02-04 17:53:00	23.15	27.2450	426.0	713.50	0.004779	1	2015	02	04 17:53:00
4	2015-02-04 17:54:00	23.15	27.2000	426.0	708.25	0.004772	1	2015	02	04 17:54:00
5	2015-02-04 17:55:00	23.10	27.2000	426.0	704.50	0.004757	1	2015	02	04 17:55:00

```
[ ] datatrain=datatrain.drop("date",axis=1)
```

```
[ ] datatrain.head()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy	Year	Month	Day
1	23.18	27.2720	426.0	721.25	0.004793	1	2015	02	04 17:51:00
2	23.15	27.2675	429.5	714.00	0.004783	1	2015	02	04 17:51:59
3	23.15	27.2450	426.0	713.50	0.004779	1	2015	02	04 17:53:00
4	23.15	27.2000	426.0	708.25	0.004772	1	2015	02	04 17:54:00
5	23.10	27.2000	426.0	704.50	0.004757	1	2015	02	04 17:55:00

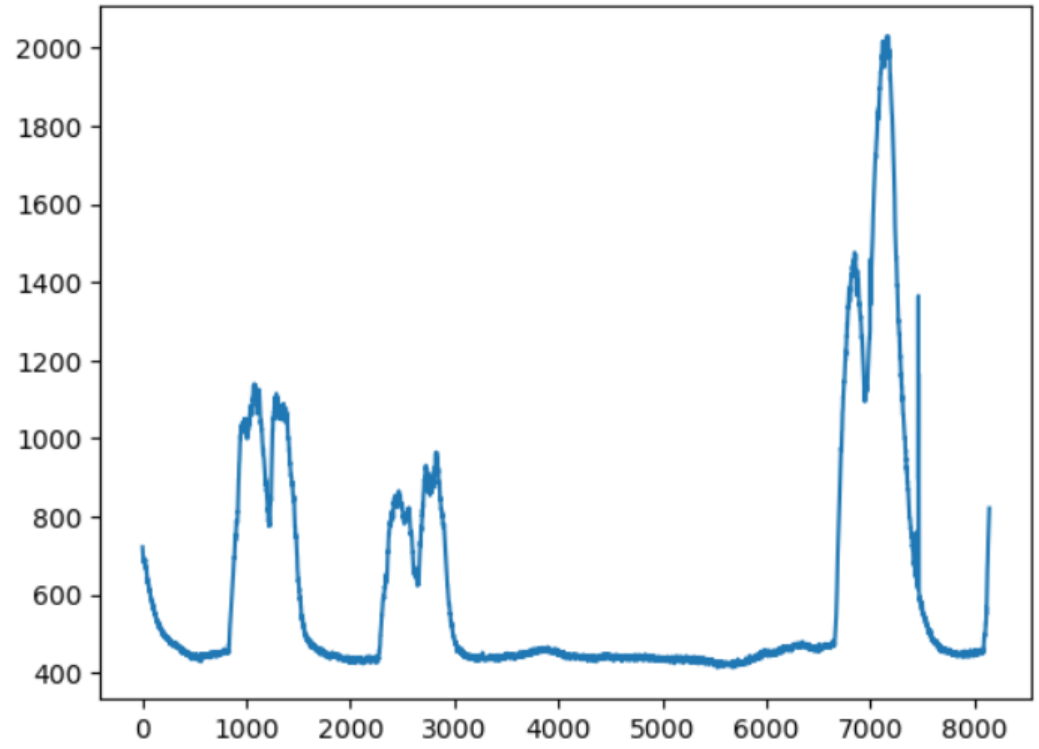
Activity 7: Use matplotlib , seaborn for visualization by using corr() values

```
[ ] datatrain.corr().Occupancy.sort_values(ascending=False)
```

```
C:\Users\Anjali Srivastava\AppData\Local\Temp\ipykernel_24904\2671749555.py:1
  datatrain.corr().Occupancy.sort_values(ascending=False)
Occupancy        1.000000
Light            0.907352
CO2              0.712235
Temperature      0.538220
HumidityRatio    0.300282
Humidity         0.132964
Name: Occupancy, dtype: float64
```

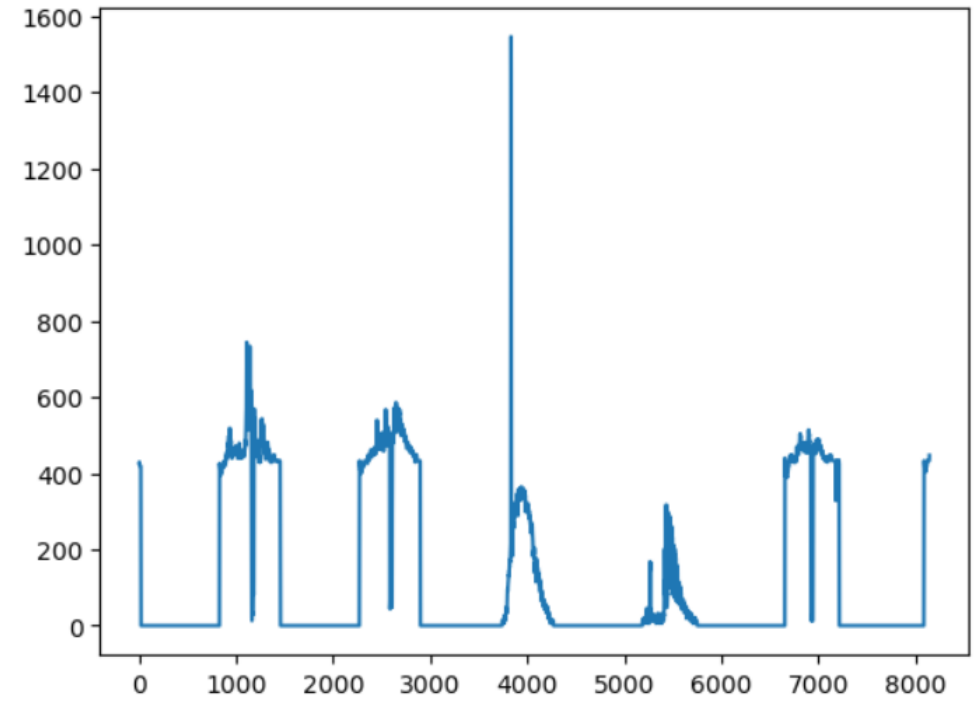
```
datatrain.CO2.plot()
```

```
<Axes: >
```



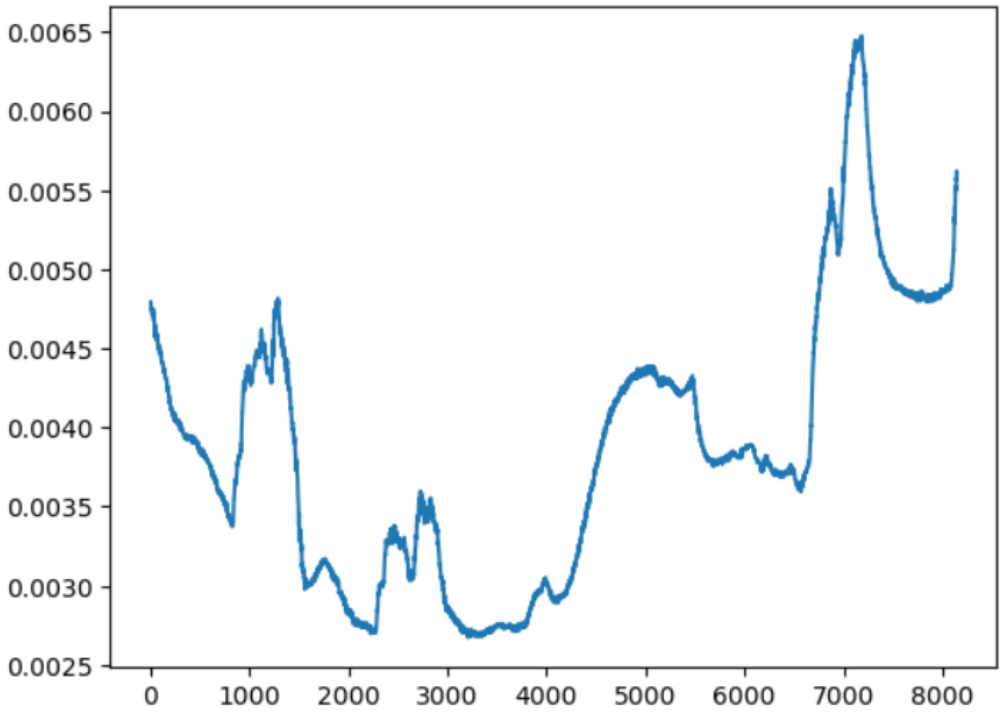

```
datatrain.Light.plot()
```

```
<Axes: >
```



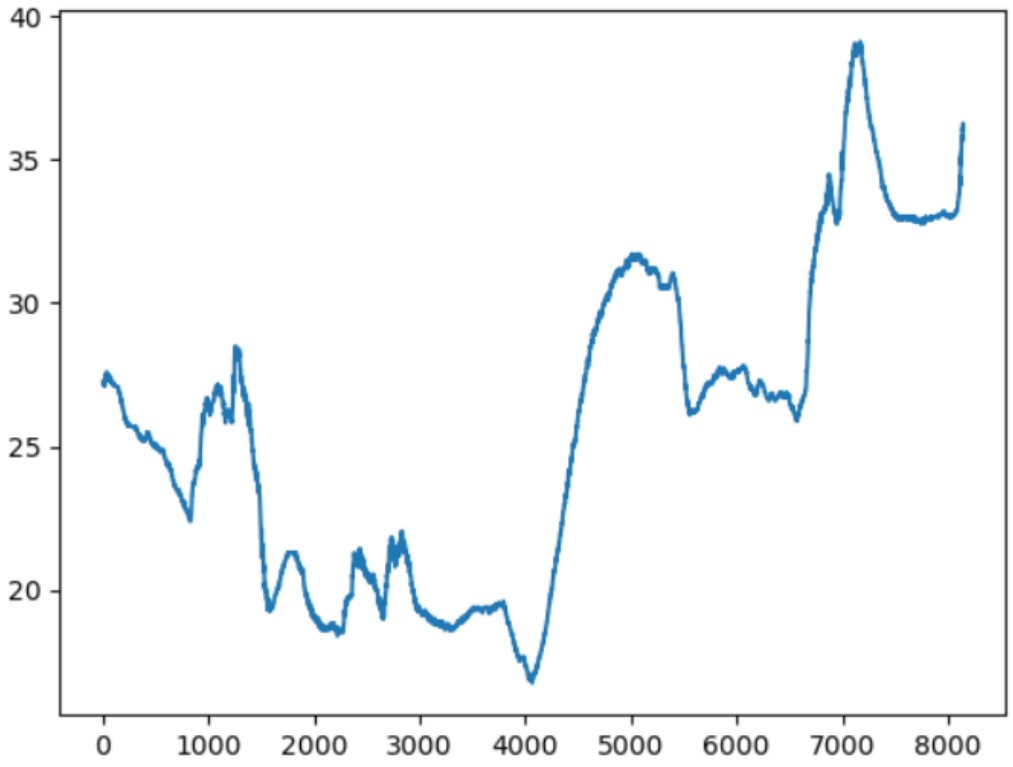
```
datatrain.HumidityRatio.plot()
```

```
<Axes: >
```



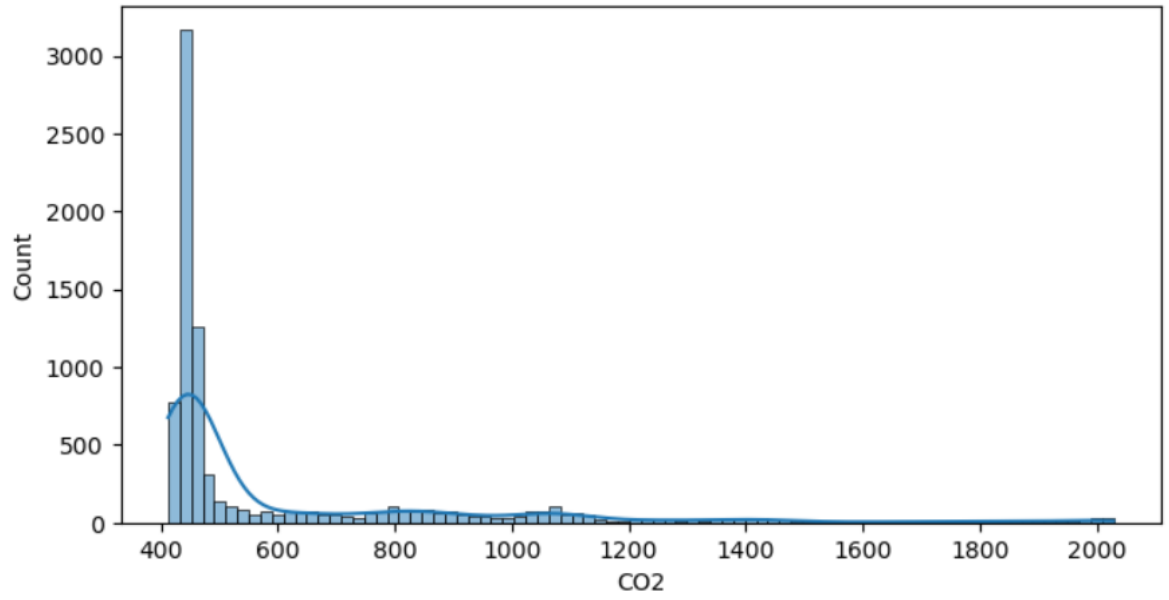

```
datatrain.Humidity.plot()
```

```
<Axes: >
```



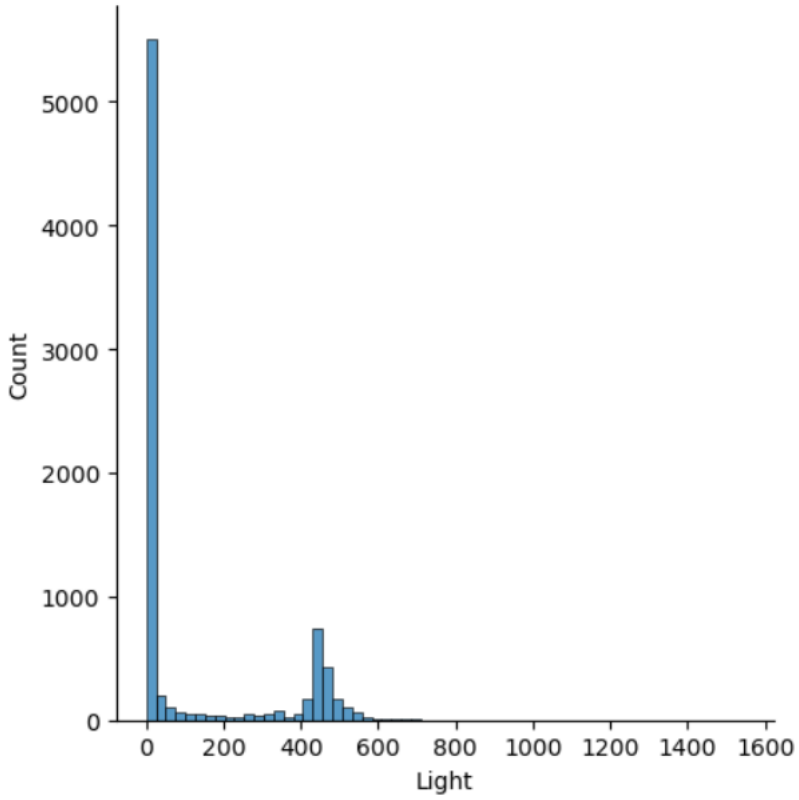
```
plt.figure(figsize=(8, 4))
sns.histplot(data=datatrain,x="CO2",kde=True)
```

```
<Axes: xlabel='CO2', ylabel='Count'>
```



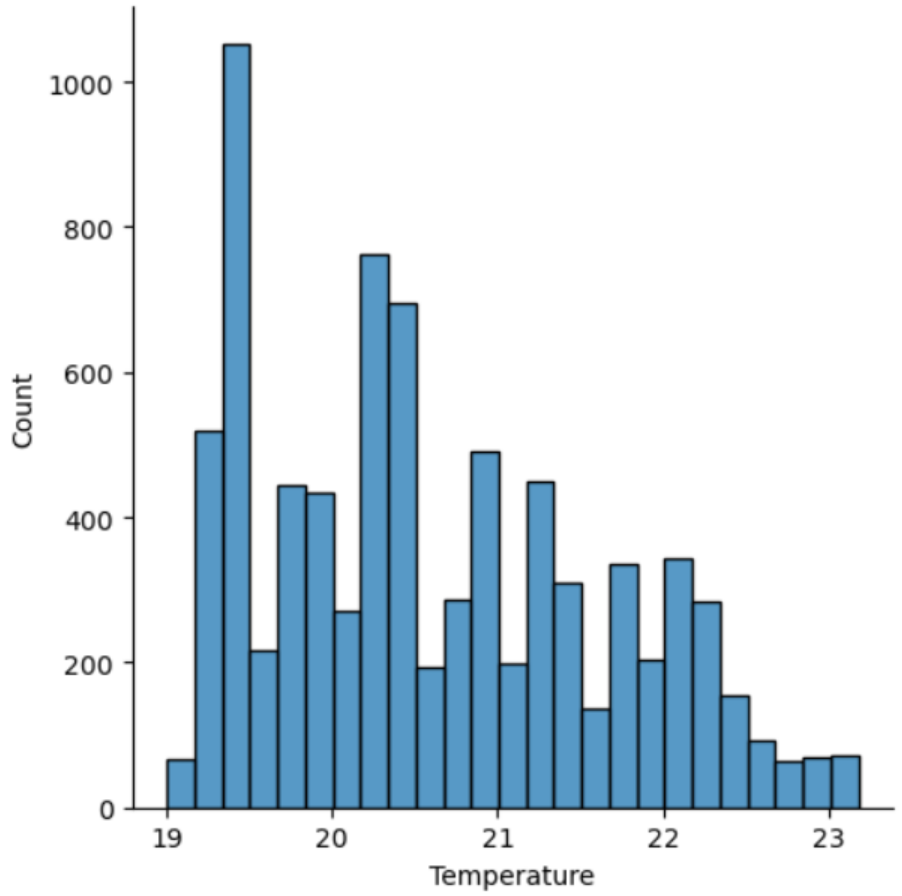

```
# plt.figure(figsize=(16,4))
sns.displot(datatrain["Light"])
```

```
<seaborn.axisgrid.FacetGrid at 0x248afe51950>
```

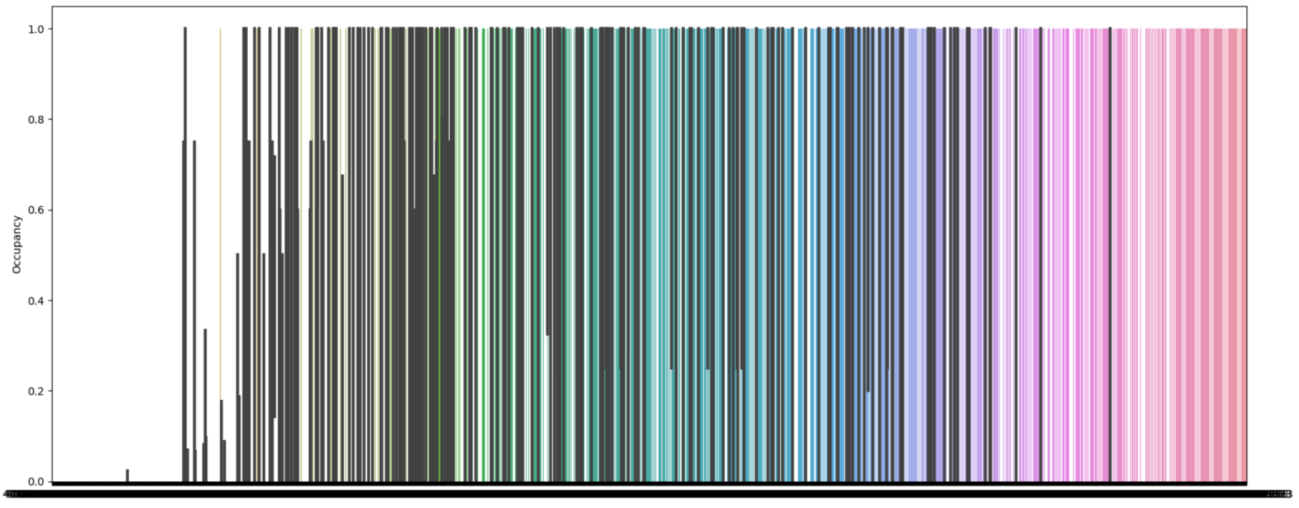


```
sns.displot(datatrain["Temperature"])
```

```
<seaborn.axisgrid.FacetGrid at 0x248afe731d0>
```

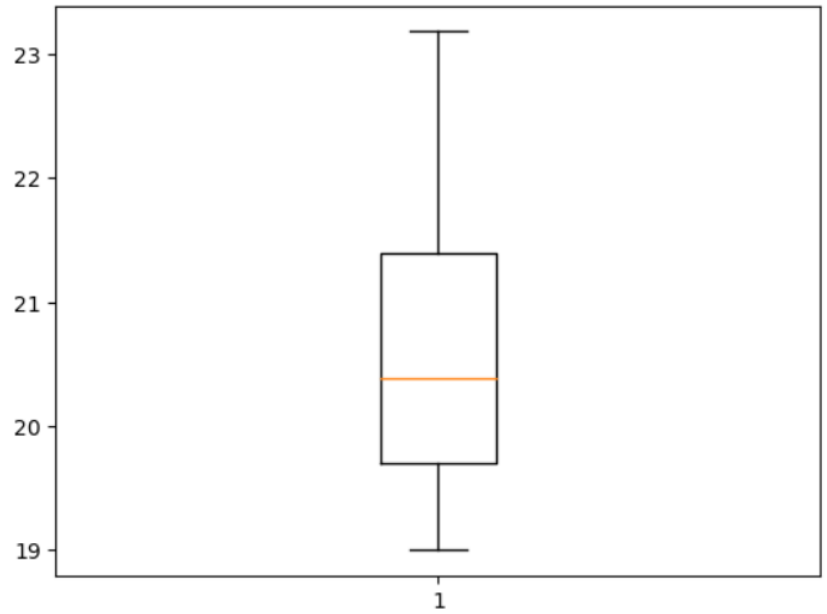



```
plt.figure(figsize=(20, 8))
sns.barplot(x=X["CO2"],y=Y)
plt.show()
```



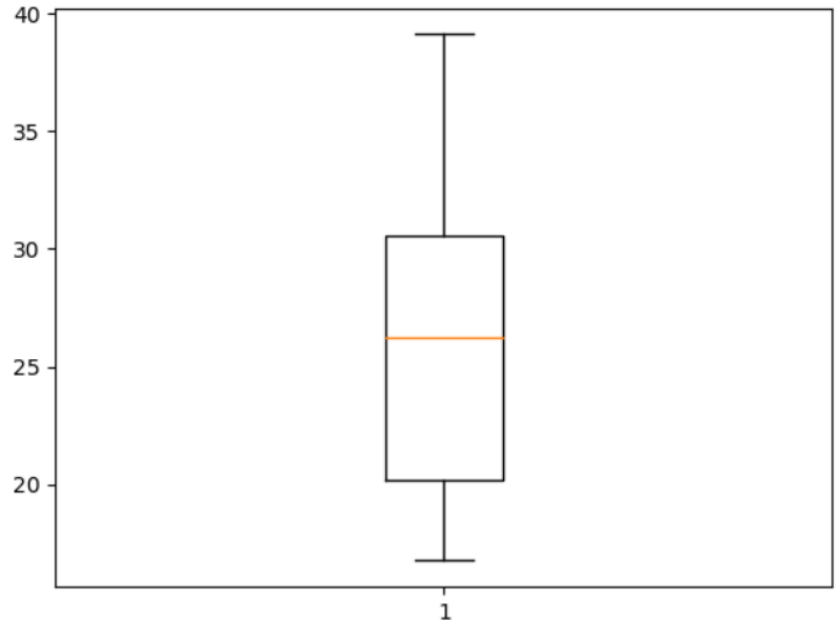
```
plt.boxplot(datatrain["Temperature"])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x248af8d6f90>,
  <matplotlib.lines.Line2D at 0x248b1062ed0>],
 'caps': [<matplotlib.lines.Line2D at 0x248b10639d0>,
  <matplotlib.lines.Line2D at 0x248b1078590>],
 'boxes': [<matplotlib.lines.Line2D at 0x248af93e750>],
 'medians': [<matplotlib.lines.Line2D at 0x248b1079050>],
 'fliers': [<matplotlib.lines.Line2D at 0x248b1078510>],
 'means': []}
```

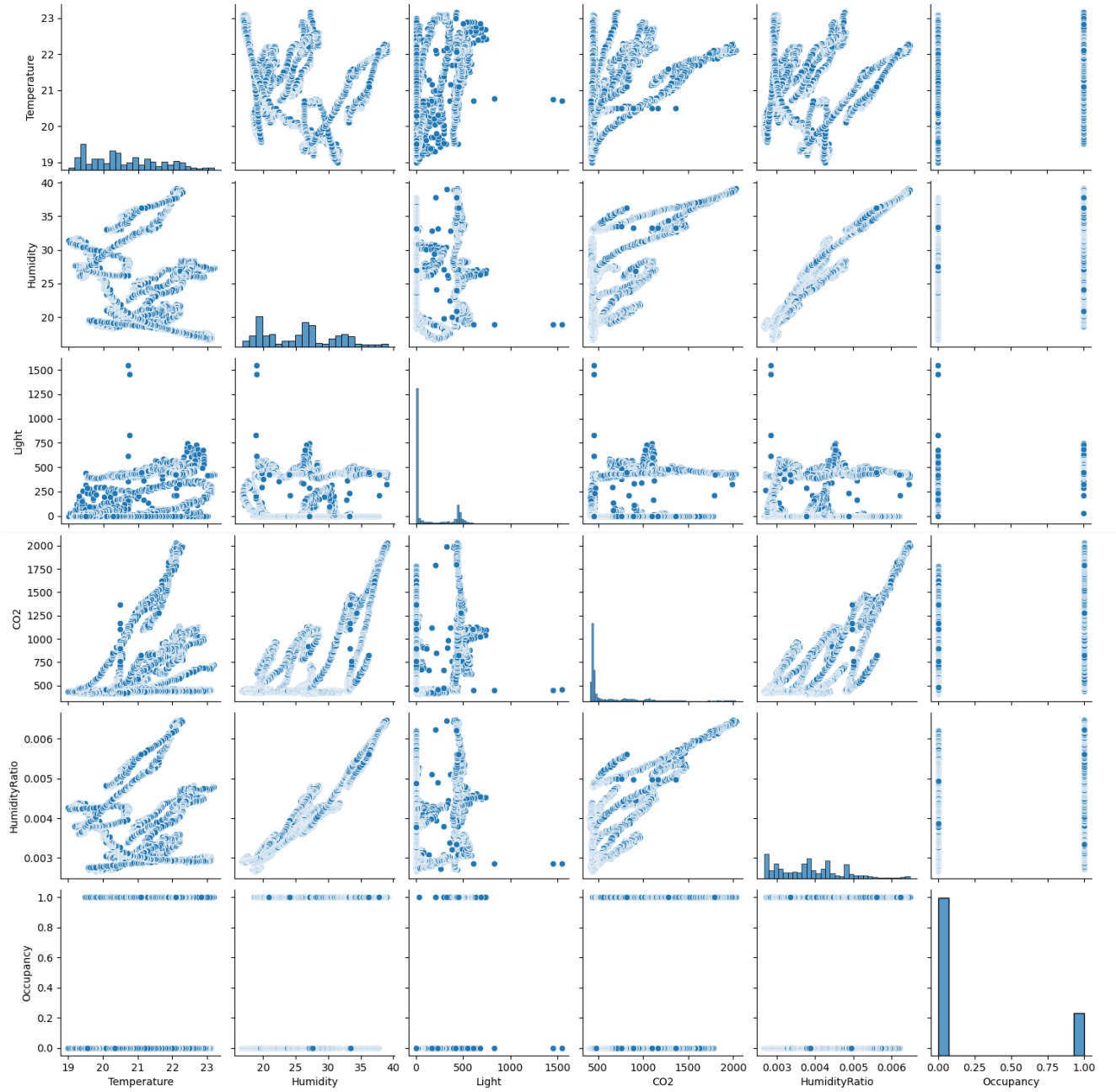



```
plt.boxplot(datatrain["Humidity"])
```

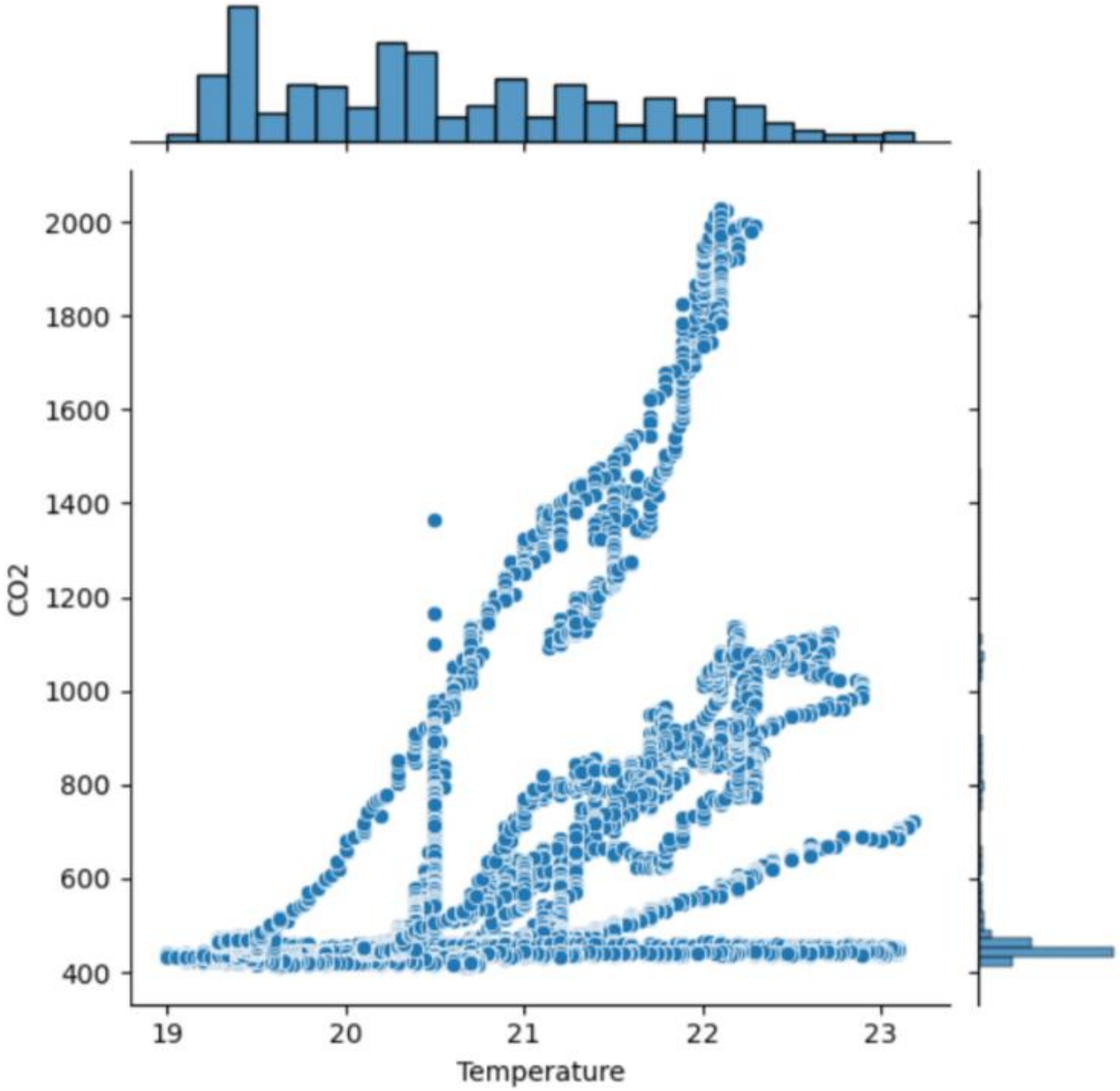
```
{'whiskers': [<matplotlib.lines.Line2D at 0x248b10c3a50>,
  <matplotlib.lines.Line2D at 0x248b10cc610>],
 'caps': [<matplotlib.lines.Line2D at 0x248b10cd190>,
  <matplotlib.lines.Line2D at 0x248b10cdcd0>],
 'boxes': [<matplotlib.lines.Line2D at 0x248b10c2fd0>],
 'medians': [<matplotlib.lines.Line2D at 0x248b10ce7d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x248b10cced0>],
 'means': []}
```



```
sns.pairplot(datatrain)
```

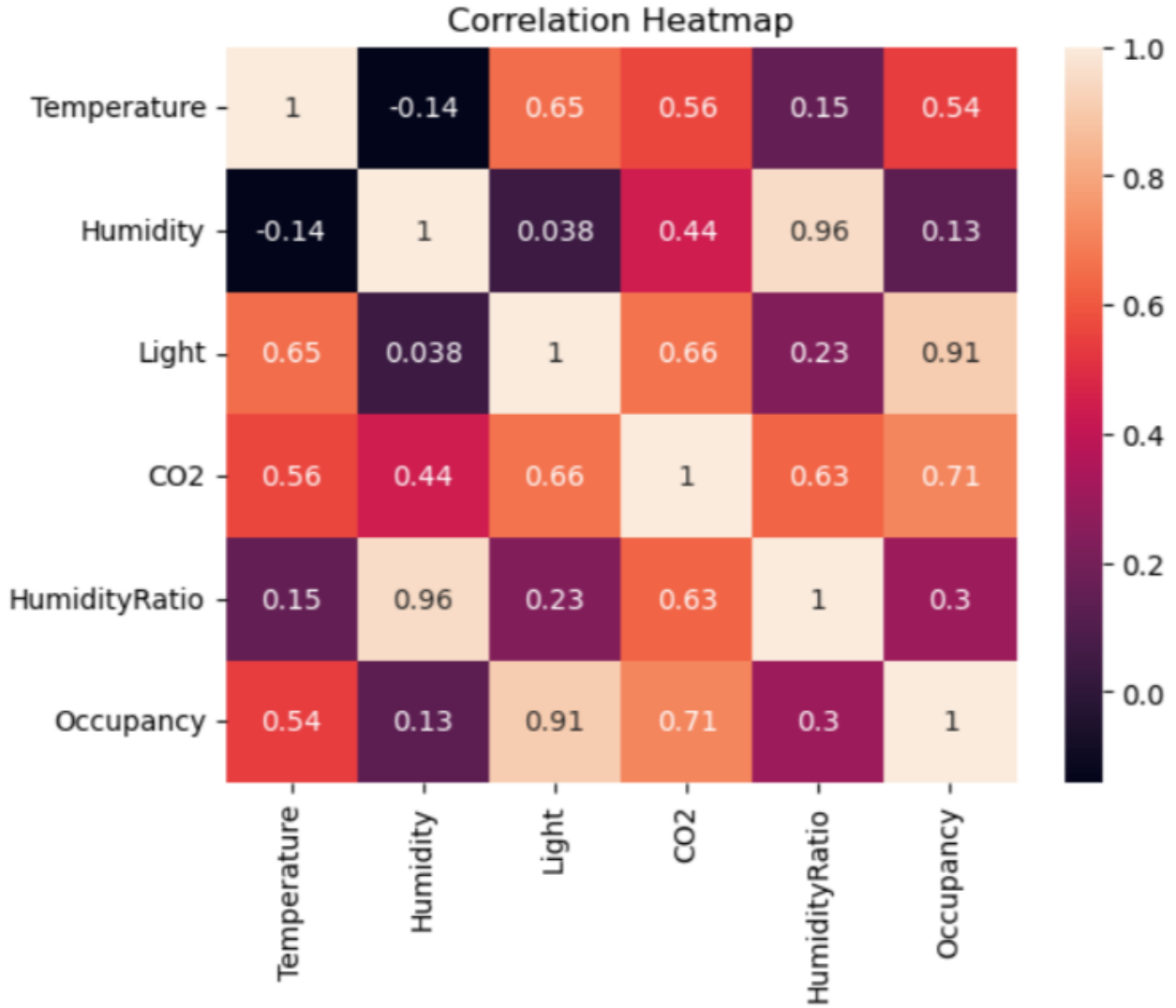


```
sns.jointplot(data=datatrain, x='Temperature', y='CO2', kind='scatter')
plt.show()
```



```
sns.heatmap(datatrain.corr(),annot=True)
plt.title("Correlation Heatmap")
```

```
C:\Users\Anjali Srivastava\AppData\Local\Temp\ipykernel_24904\1571223003.py:1:
  sns.heatmap(datatrain.corr(),annot=True)
Text(0.5, 1.0, 'Correlation Heatmap')
```



Activity 8: split into x and y

X and Y split here =>(dependent and independent)

```
[ ]  Y=datatrain["Occupancy"]
     X=datatrain.drop("Occupancy",axis=1)
```

```
[ ]  Y.head()
```

```
1    1
2    1
3    1
4    1
5    1
Name: Occupancy, dtype: int64
```

```
[ ]  X.head()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Year	Month	Day
1	23.18	27.2720	426.0	721.25	0.004793	2015	02	04 17:51:00
2	23.15	27.2675	429.5	714.00	0.004783	2015	02	04 17:51:59
3	23.15	27.2450	426.0	713.50	0.004779	2015	02	04 17:53:00
4	23.15	27.2000	426.0	708.25	0.004772	2015	02	04 17:54:00
5	23.10	27.2000	426.0	704.50	0.004757	2015	02	04 17:55:00

Activity 9: check for the shape of x & y

```
[ ]  X.shape
```

```
(8143, 8)
```

```
[ ]  Y.shape
```

```
(8143,)
```

Activity 10: Use LabelEncoder for converting the obj into int

```
le1=LabelEncoder()
le2=LabelEncoder()
le3=LabelEncoder()
```

```
X["Year"]=le1.fit_transform(datatrain.Year)
X["Month"]=le2.fit_transform(datatrain.Month)
X["Day"]=le3.fit_transform(datatrain.Day)
```

```
X.head()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Year	Month	Day
1	23.18	27.2720	426.0	721.25	0.004793	0	0	0
2	23.15	27.2675	429.5	714.00	0.004783	0	0	1
3	23.15	27.2450	426.0	713.50	0.004779	0	0	2
4	23.15	27.2000	426.0	708.25	0.004772	0	0	3
5	23.10	27.2000	426.0	704.50	0.004757	0	0	4

Activity 11: train x & y

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

```
X_train.shape
```

```
(6514, 8)
```

```
X_test.shape
```

```
(1629, 8)
```

```
Y_train.shape
```

```
(6514,)
```

Activity 12: Use Standard scaler and perform algorithm

```
sc=StandardScaler()
```

```
X_train=sc.fit_transform(X_train)
```

```
X_test=sc.transform(X_test)
```

```
from sklearn.tree import DecisionTreeClassifier
DC=DecisionTreeClassifier(random_state=0)
DC.fit(X_train,Y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
X_test_predict=DC.predict(X_test)
```

```
X_test_predict
```

```
array([1, 1, 0, ..., 0, 0, 0], dtype=int64)
```

Activity 13: check for accuracy

```
from sklearn.metrics import accuracy_score
```

```
acc=accuracy_score(Y_test,X_test_predict)
```

```
acc
```

```
0.9914057704112953
```

Activity 14: check for confusion metrics

```
cm=confusion_matrix(Y_test,X_test_predict)
```

```
cm   #FN   #FP
     #TN   #TP
```

```
array([[1255,    9],
       [   5,  360]], dtype=int64)
```

```
import sklearn.metrics as mat
```

```
fpr,tpr,threshold = mat.roc_curve(Y_test,X_test_predict)
```

```
roc_auc=mat.auc(fpr,tpr)
```

```
fpr
```

```
array([0.        , 0.00712025, 1.        ])
```

```
tpr
```

```
array([0.        , 0.98630137, 1.        ])
```

```
threshold
```

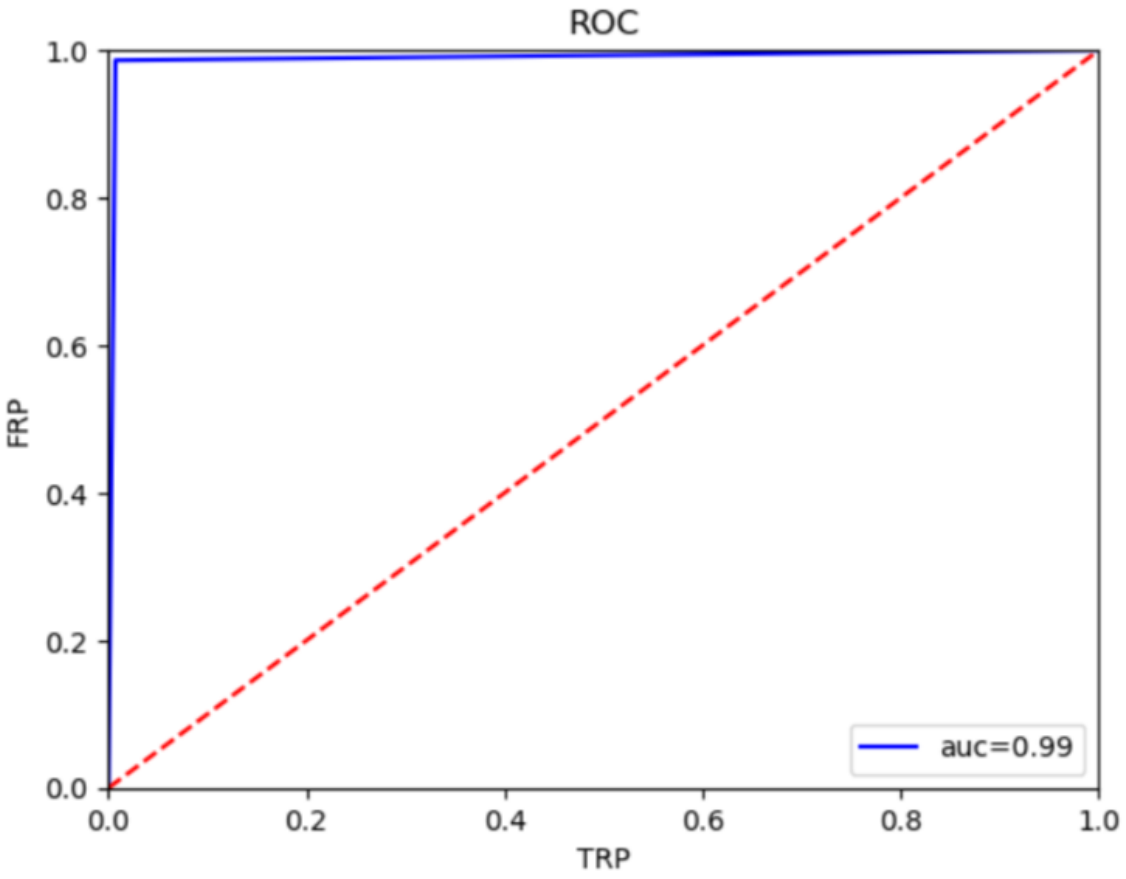
```
array([inf,  1.,  0.])
```

```
roc_auc
```

```
0.9895905583492283
```

Activity 15: check the graph

```
plt.title('ROC')
plt.plot(fpr,tpr,"b",label="auc=%0.2f"%roc_auc)
plt.legend(loc="lower right")
plt.plot([0,1],[0,1],'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.xlabel('TRP')
plt.ylabel('FRP')
plt.show()
```



Activity 16: Make predictions using your own data

```
y1=DC.predict([[23.18,27.2720,426.0,721.25,0.004793,0,0,0]])
```

```
y1
if(y1==1):
    print("OCCUPANCY THERE")
else:
    print("OCCUPANCY NOT THERE")
```

```
OCCUPANCY THERE
```

```
y2=DC.predict([[21.50,26.1000,0.000,500.00,0.004137,0,0,200]])
y2
if(y2==1):
    print("OCCUPANCY THERE")
else:
    print("OCCUPANCY NOT THERE")
```

```
OCCUPANCY NOT THERE
```

Activity 17: Do the pickle

```
import pickle
```

```
pickle.dump(DC,open('occupancy.pkl','wb'))
```

MODEL DEPLOYMENT

The deployment process involves creating a Flask website where the machine learning model is integrated using the "pickle" library. Once the model is trained and saved as a pickle file, it can be loaded within a Flask web application. Users can interact with the model by inputting data through the website, and the model's predictions can be displayed in real-time. Here is the link for the demonstration of our website doing real-time predictions using our ML model:

https://drive.google.com/file/d/1hqLtsiglvR0faYwl-VdosWSQTrbsmsW-/view?usp=sharing

CONCLUSION

This forward-thinking solution, combining environmental factors with occupancy data to predict hotel occupancy and demand, stands as a novel approach to enhancing the hospitality industry. In a similar vein, the study's exploration of machine learning techniques serves as a valuable precursor. The study compared various models, including Ridge Regression, Kernel Ridge Regression, Multilayer Perceptron, and Radial Basis Function Networks, to advance the field of occupancy prediction. Three distinct datasets, encompassing time series data, time series data supplemented with additional variables, and reservations data, were meticulously crafted to facilitate model training and validation.

Grid search played a pivotal role in identifying optimal parameters for these models. The research findings underscore the efficacy of the Ridge regression model, particularly when equipped with quadratic features and trained on the reservations dataset. This model showcased superior performance, boasting a validation set Mean Absolute Percentage Error (MAPE) of 8.2012% and a test set MAPE of 8.6561%, all while avoiding overfitting. Notably, models trained on data enriched with additional variables exhibited a modest but notable performance boost, showcasing the power of contextual information integration.

The study emphasized the advantages of utilizing bookings and reservations known in advance for occupancy prediction, ultimately yielding the most promising results. These insights are not only promising but also strongly advocate for the utilization of black-box machine learning tools in the estimation of hotel occupancy. These tools are designed to minimize the need for advanced statistical expertise among hotel staff, thereby facilitating a more efficient application of Revenue Management techniques within the hospitality sector. Combining these findings with the innovative environmental-based solution offers a holistic and forward-thinking approach to revolutionizing the field of occupancy and demand prediction in the hospitality industry.