DSE 309 – Advanced Programming in Python

Final Project

Occupancy Prediction using environmental observations

Name – Rishabh Uikey

Roll no – 18200

Introduction

As we all know, energy saving has become one of the significant aspects to consider while deriving anything new or even making existing things more effective and efficient. Many day-to-day techs like Light Bulbs, Air conditioners, heaters, etc., run continuously and consume energy even when not in use. So here, in this project, we are trying to predict particular room occupancy- i.e., whether the room is occupied or empty. By doing so, we can say that if there's no one in the room- switch off the lights, fan, AC, etc.

About the Dataset

Here we are taking Room's environments factor (Like Light intensity, CO2 level, Temperature, Humidity, and Humidity ratio.) these data have been collected by different sensors, which help us predict the room's occupancy status. We are taking the dataset from the UCI ML repository. For prediction, we are using machine learning algorithms instead of physical sensors as sensors will require time to time maintenance, and ML algorithm will be cost and maintenance-free.

Analysis

We are taking the dataset from the UCL ML repository and cleaning it to improve the data quality as it increases overall productivity.

We first reset the index so that handling becomes easier. Describe () function gives the details regarding the mean, no of count, standard deviation of whole dataset.

Python Final Project - Occupancy Estimation- Prediction using environmental observations

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import math
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, mean_squared_error, mean_absolute_error, f1_
from sklearn.svm import SVC
```

In [2]:
```
# load data
dataset= pd.read_csv(r'D:\books\4th year 1st sem books\python\final project\main\dataset.csv')
dataset
```

Out[2]:

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	23.700000	26.272000	585.200000	749.200000	0.004764	1
1	23.718000	26.290000	578.400000	760.400000	0.004773	1
2	23.730000	26.230000	572.666667	769.666667	0.004765	1
3	23.722500	26.125000	493.750000	774.750000	0.004744	1
4	23.754000	26.200000	488.600000	779.000000	0.004767	1
...	...	...	...	...	...	...
2660	24.290000	25.700000	808.000000	1150.250000	0.004829	1
2661	24.330000	25.736000	809.800000	1129.200000	0.004848	1
2662	24.330000	25.700000	817.000000	1125.800000	0.004841	1
2663	24.356667	25.700000	813.000000	1123.000000	0.004849	1
2664	24.408333	25.681667	798.000000	1124.000000	0.004860	1

2665 rows × 6 columns

In [3]:
```
dataset.drop(dataset.columns[dataset.columns.str.contains('unnamed',case = False)],axis = 1, inplace = True)
```

In [4]:
```
dataset
```

Out[4]:

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	23.700000	26.272000	585.200000	749.200000	0.004764	1
1	23.718000	26.290000	578.400000	760.400000	0.004773	1
2	23.730000	26.230000	572.666667	769.666667	0.004765	1
3	23.722500	26.125000	493.750000	774.750000	0.004744	1
4	23.754000	26.200000	488.600000	779.000000	0.004767	1
...	...	...	...	...	...	...
2660	24.290000	25.700000	808.000000	1150.250000	0.004829	1
2661	24.330000	25.736000	809.800000	1129.200000	0.004848	1
2662	24.330000	25.700000	817.000000	1125.800000	0.004841	1
2663	24.356667	25.700000	813.000000	1123.000000	0.004849	1
2664	24.408333	25.681667	798.000000	1124.000000	0.004860	1

2665 rows × 6 columns

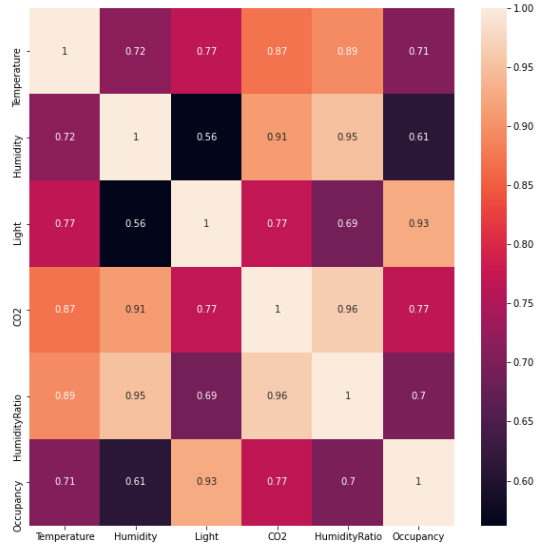
In [5]:
```
dataset.describe()
```

Out[5]:

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	2665.000000	2665.000000	2665.000000	2665.000000	2665.000000	2665.000000
mean	21.433876	25.353937	193.227556	717.906470	0.004027	0.364728
std	1.028024	2.436842	250.210906	292.681718	0.000611	0.481444
min	20.200000	22.100000	0.000000	427.500000	0.003303	0.000000
25%	20.650000	23.260000	0.000000	466.000000	0.003529	0.000000
50%	20.890000	25.000000	0.000000	580.500000	0.003815	0.000000
75%	22.356667	26.856667	442.500000	956.333333	0.004532	1.000000
max	24.408333	31.472500	1697.250000	1402.250000	0.005378	1.000000

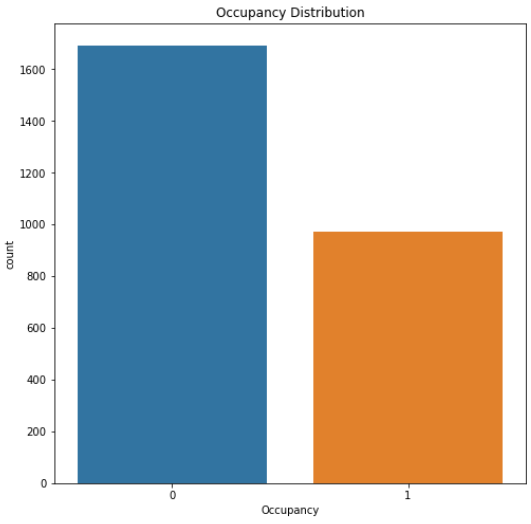
In [6]:
```
dataset.reset_index(drop=True,inplace=True)
```

```
In [7]: plt.figure(figsize=(10,10))
        sns.heatmap(dataset.corr(),annot=True)
Out[7]: <AxesSubplot:>
```



The above plot is the correlation plot. It helps us to find the dependency between multiple variables. The value-1 denotes the perfect correlation and value-0 denotes no correlation between variable.

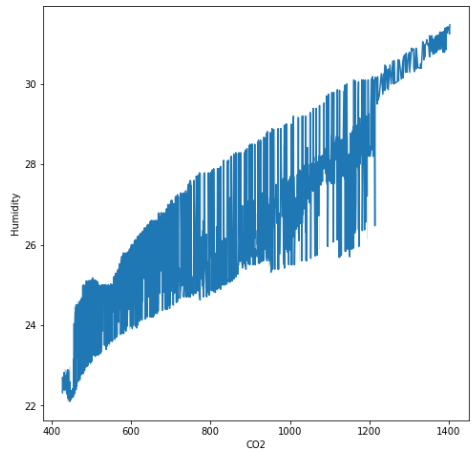
```
In [44]: plt.figure(figsize=(8,8))
         sns.countplot(x='Occupancy',data=dataset)
         plt.title("Occupancy Distribution")
Out[44]: Text(0.5, 1.0, 'Occupancy Distribution')
```



Above plot is the count-plot it simply counts the number of "**0**" and number of "**1**" in the dataset.

```
In [54]: # As we can see form the correlation map, there's a huge relation between occupancy and light also between CO2 and humidity
         fig = plt.figure(figsize=(8, 8))
         sns.lineplot(x="CO2",y="Humidity", data=dataset)

Out[54]: <AxesSubplot:xlabel='CO2', ylabel='Humidity'>
```

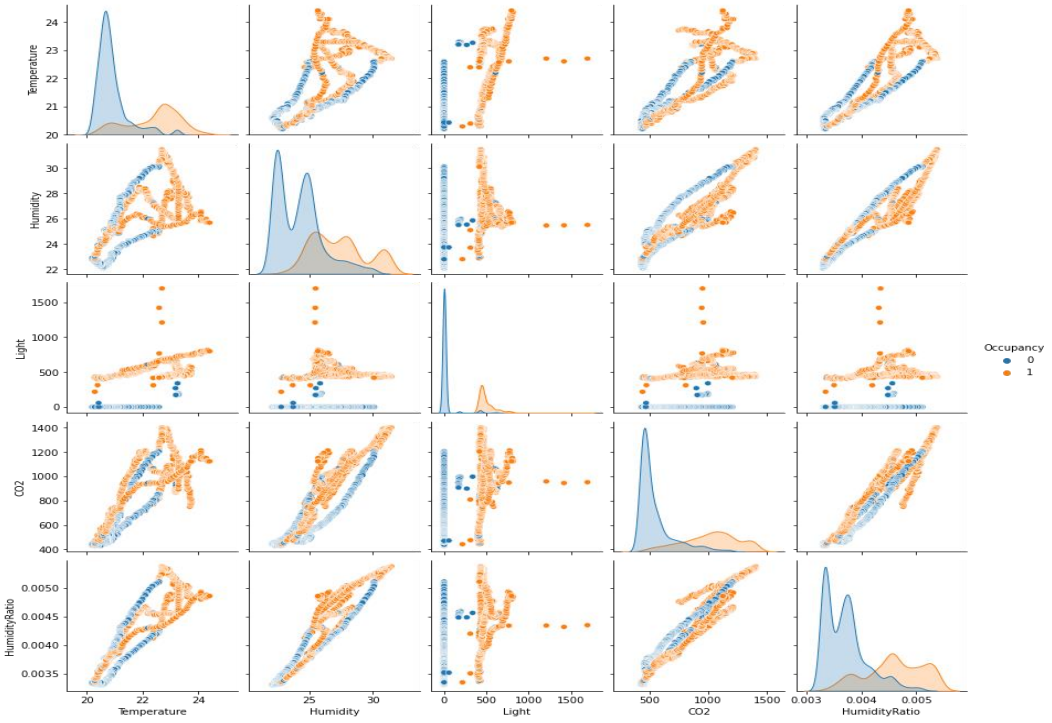


Above line is the line plot between the CO2 data and Humidity data over the entire dataset.

```
In [9]: plt.figure(figsize=(10,6),dpi = 150)
        sns.pairplot(dataset , hue = 'Occupancy')

Out[9]: <seaborn.axisgrid.PairGrid at 0x22643f017f0>

        <Figure size 1500x900 with 0 Axes>
```



The diagonal line represents the density plot between the respective columns. While in other plot linear positive progression represents positive correlation, while negative progression represents negative correlation.

Here we are normalizing the data i.e., resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1.

In second step, we are splitting the data into training and testing data for both x and y. And calculating its size.

```
In [10]: X=dataset.drop('Occupancy', axis=1)
         y=dataset['Occupancy']

In [11]: # StandardScaler is to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation

         scale=StandardScaler()
         X=scale.fit_transform(X)
         X

Out[11]: array([[2.20476235, 0.37681367, 1.56686218, 0.10694007, 1.20754022],
                [2.22227495, 0.38420166, 1.53968001, 0.14521407, 1.22146086],
                [2.23395002, 0.35957501, 1.51676171, 0.17688126, 1.20916118],
                ...,
                [2.81770341, 0.14203962, 2.4934545 , 1.39390353, 1.33301272],
                [2.84364801, 0.14203962, 2.47746498, 1.38433502, 1.34579091],
                [2.89391566, 0.13451481, 2.41750431, 1.38775235, 1.36487235]])

In [12]: # Splitting the data
         X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33, shuffle=False)

         print(X_train.shape);
         print(X_test.shape);
         print(y_train.shape);
         print(y_test.shape);

         (1785, 5)
         (880, 5)
         (1785,)
         (880,)
```

For classification

We are using different model to find the best fitting model for predicting the occupancy.

Here we are using 3 different model for classification: -

1) SVM (Support Vector Machine)
2) Decision Tree
3) KNN (K-Nearest Neighbours)

SVM (Support Vector Machine)

The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

SVM Model

In [13]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
print("Accuracy for SVM on testing data: {}%".format(round((svm_model.score(X_test, y_test)*100),2)))
```

```
Accuracy for SVM on testing data: 97.84%
```

In [14]:
```
y_predict = svm_model.predict(X_test)
print(y_predict);
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

In [15]:
```
#Actual value and the predicted value
difference1 = pd.DataFrame({'Actual value(temperature)': y_test, 'Predicted value(temperature)':y_predict})
difference1
```


In [15]:
```
#Actual value and the predicted value
difference1 = pd.DataFrame({'Actual value(temperature)': y_test, 'Predicted value(temperature)':y_predict})
difference1
```

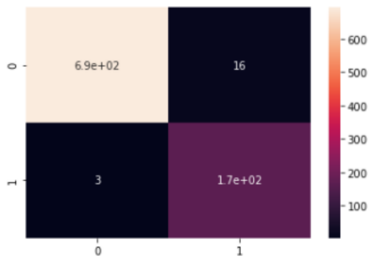

Out[15]:

	Actual value(temperature)	Predicted value(temperature)
1785	0	0
1786	0	0
1787	0	0
1788	0	0
1789	0	0
...	...	...
2660	1	1
2661	1	1
2662	1	1
2663	1	1
2664	1	1

880 rows × 2 columns

In [16]:
```
#Calculating Confusion Matrix
CM1 = confusion_matrix(y_test, y_predict)
sns.heatmap(CM1,annot=True)
print('Confusion Matrix is : \n', CM1)
```

```
Confusion Matrix is :
 [[694  16]
 [  3 167]]
```



Using the SVM model we predicted the occupancy count over the testing data. And we printed the confusion matrix to have the estimate for the correct prediction and False prediction value no.

```
In [17]: #Calculating classification Report :
         Report = classification_report(y_test,y_predict)
         print(Report)
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       710
           1       0.91      0.98      0.95       170

    accuracy                           0.98       880
   macro avg       0.95      0.98      0.97       880
weighted avg       0.98      0.98      0.98       880
```

```
In [19]: # calculating mean squared error and root mean sqared error

         mse=mean_squared_error(y_test,y_predict)
         rms=math.sqrt(mse)
         print("Mean Squared Error :",mse)
         print("Root Mean Squared Error :",rms)
```

```
Mean Squared Error : 0.02159090909090909
Root Mean Squared Error : 0.14693845341131467
```

The above table shows the classification report for the SVM model. This report includes precision, recall and f1-score and its accuracy.

Mean square error for SVM model = 0.02159

Root mean squared error for SVM model= 0.1469

Decision Tree

It is a tree-structured classifier, where internal nodes represent the features of the dataset, branches represent the decision rules and each leaf node represents the outcome. In decision tree, there are two nodes, which are the decision node and leaf node. It is a graphical representation for getting all the possible solutions to a decision based on the condition. It is called tree because it functions similar to a tree, it starts with the root node, which expands on further branches.

Decision Tree

In [20]:
```python
from sklearn.tree import DecisionTreeClassifier

classifier= DecisionTreeClassifier(criterion='entropy')
classifier.fit(X_train, y_train)
```

Out[20]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [21]:
```python
classifier.score(X_test,y_test)
y_predict1 = classifier.predict(X_test)
print(y_predict1);
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

In [22]:
```python
#Actual value and the predicted value
diff = pd.DataFrame({'Actual value(temperature)': y_test, 'Predicted value(temperature)':y_predict1})
diff
```

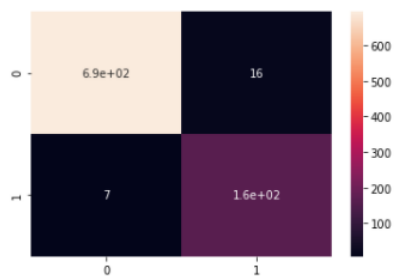
Out[22]:

	Actual value(temperature)	Predicted value(temperature)
1785	0	0
1786	0	0
1787	0	0
1788	0	0
1789	0	0
...	...	...
2660	1	1
2661	1	1
2662	1	1
2663	1	1
2664	1	1

880 rows × 2 columns

In [23]:
```python
#Calculating Confusion Matrix
CM2 = confusion_matrix(y_test, y_predict1)
sns.heatmap(CM2,annot=True)
print('Confusion Matrix is : \n', CM2)
```

```
Confusion Matrix is :
 [[694  16]
 [  7 163]]
```



Using the Decision tree model, we predicted the occupancy count over the testing data. And we printed the confusion matrix to have the estimate for the correct prediction and False prediction value no.

In [24]:
```
#Calculating classification Report :
Report1 = classification_report(y_test,y_predict1)
print(Report1)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       710
           1       0.91      0.96      0.93       170

    accuracy                           0.97       880
   macro avg       0.95      0.97      0.96       880
weighted avg       0.97      0.97      0.97       880
```

In [25]:
```
# calculating mean squared error and root mean sqared error

mse1=mean_squared_error(y_test,y_predict1)
rms1=math.sqrt(mse1)
print("Mean Squared Error :",mse1)
print("Root Mean Squared Error :",rms1)
```

```
Mean Squared Error : 0.026136363636363635
Root Mean Squared Error : 0.1616674476707158
```

The above table shows the classification report for the Decision Tree model. This report includes precision, recall and f1-score and its accuracy.

Mean square error for Decision Tree model = 0.02613

Root mean squared error for Decision Tree model= 0.1616

KNN (K-Nearest Neighbours)

KNN is one of the simplest ML algo based on supervised learning technique. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well-suited category by using KNN algorithm. It's a non-parametric algorithm which means it doesn't make any assumption on underlying data. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

KNN (K- Nearest Neighbors)

In [29]:
```
#  trying different hyperparameters on KNN model
# for n= 1,5,10

# for n=1

knn_model = KNeighborsClassifier(n_neighbors=1)
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
print(knn_model.score(X_test,y_test));
```

```
0.9738636363636364
```

In [37]:
```
# for n=5

knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred1 = knn_model.predict(X_test)
print(knn_model.score(X_test,y_test));
```

```
0.9806818181818182
```

```
In [38]: # for n=10

         knn_model = KNeighborsClassifier(n_neighbors=10)
         knn_model.fit(X_train, y_train)
         y_pred2 = knn_model.predict(X_test)
         print(knn_model.score(X_test,y_test));

         0.9818181818181818

In [39]: print(y_pred2);

         [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Here we are using 3 different fold values. For n=1 i.e., means there will be only one-fold. For k=1 model score is about 97 %. For k=5, there will be 5 folds few folds will be used testing while other will be used for testing. Model score is about 98.068%. For k=10, there will be 10 folds, model score is about 98.1818%. and using these models we predict the occupancy count over the testing data. And to printed the confusion matrix to have the estimate for the correct prediction and False prediction value no.

```
In [40]: #Actual value and the predicted value
         difference = pd.DataFrame({'Actual value(temperature)': y_test, 'Predicted value(temperature)':y_pred2})
         difference
```

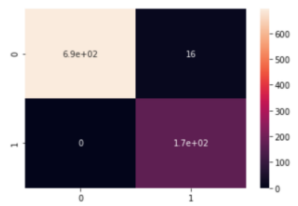
Out[40]:

	Actual value(temperature)	Predicted value(temperature)
1785	0	0
1786	0	0
1787	0	0
1788	0	0
1789	0	0
...	...	...
2660	1	1
2661	1	1
2662	1	1
2663	1	1
2664	1	1

880 rows × 2 columns

```
In [41]: #Calculating Confusion Matrix
         CM = confusion_matrix(y_test, y_pred2)
         sns.heatmap(CM,annot=True)
         print('Confusion Matrix is : \n', CM)

         Confusion Matrix is :
          [[694  16]
          [  0 170]]
```



Using the KNN model, we predicted the occupancy count over the testing data. And we printed the confusion matrix to have the estimate for the correct prediction and False prediction value no.

```
In [42]: #Calculating classification Report :
         Report3 = classification_report(y_test,y_pred2)
         print(Report3)
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       710
           1       0.91      1.00      0.96       170

    accuracy                           0.98       880
   macro avg       0.96      0.99      0.97       880
weighted avg       0.98      0.98      0.98       880
```

```
In [43]: # calculating mean squared error and root mean sqared error

         mse2=mean_squared_error(y_test,y_pred2)
         rms2=math.sqrt(mse2)
         print("Mean Squared Error :",mse2)
         print("Root Mean Squared Error :",rms2)
```

```
Mean Squared Error : 0.01818181818181818
Root Mean Squared Error : 0.13483997249264842
```

The above table shows the classification report for the KNN model. This report includes precision, recall and f1-score and its accuracy.

Mean square error for KNN model = 0.01818

Root mean squared error for KNN model= 0.1348

Result & Conclusion

From the 3 different models i.e., SVM, Decision Tree and KNN model by comparing their model score and there mean squared error and root mean squared error we can say that KNN model performs the best as it has accuracy of about 98%. Hence, we can state that KNN model is the best for predicting the Occupancy count.

References

- M. Azam, M. Blayo, J. Venne and M. Allegue-Martinez, "Occupancy Estimation Using Wifi Motion Detection via Supervised Machine Learning Algorithms," 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2019, pp. 1-5, doi: 10.1109/GlobalSIP45357.2019.8969297.

- A. P. Singh, V. Jain, S. Chaudhari, F. A. Kraemer, S. Werner and V. Garg, "Machine Learning-Based Occupancy Estimation Using Multivariate Sensor Nodes," 2018 IEEE Globecom Workshops (GC Wkshps), 2018, pp. 1-6, doi: 10.1109/GLOCOMW.2018.8644432.
- Kaggle dataset