INTRODUCTION TO DATA SCIENCE

Room Occupancy Estimation

Project Report By

Ronit Gupta(21UCS173)

Saumya Saraswat(21UCS184)

Prajwal Sugandhi(21UCS152)

Nayan Goyal(21UCS137)

Course Instructors

Dr. Lal Upendra Pratap Singh

Dr. Subrat Dash

Dr. Aloke Datta

Table Of Contents

Problem Statement

We must choose a dataset from the specified website and carry out the subsequent actions:

1. Pre-processing and visualising data

2. Describe each inference we drew from the data.

3. Describe the ML Classification Algorithms being used and why they are used.

4. Implementing those algorithms

5. Provide a visual representation of the testing set's outcome.

All of the work is done with the assistance of pre-existing Python libraries, like:

- scikit_learn
- matplotlib
- seaborn
- numpy
- pandas

Introduction To The Dataset

This dataset is related to room occupancy estimation. The goal is to estimate the precise number of occupants in a room using multiple non-intrusive environmental sensors like temperature, light, sound, CO2, and PIR.

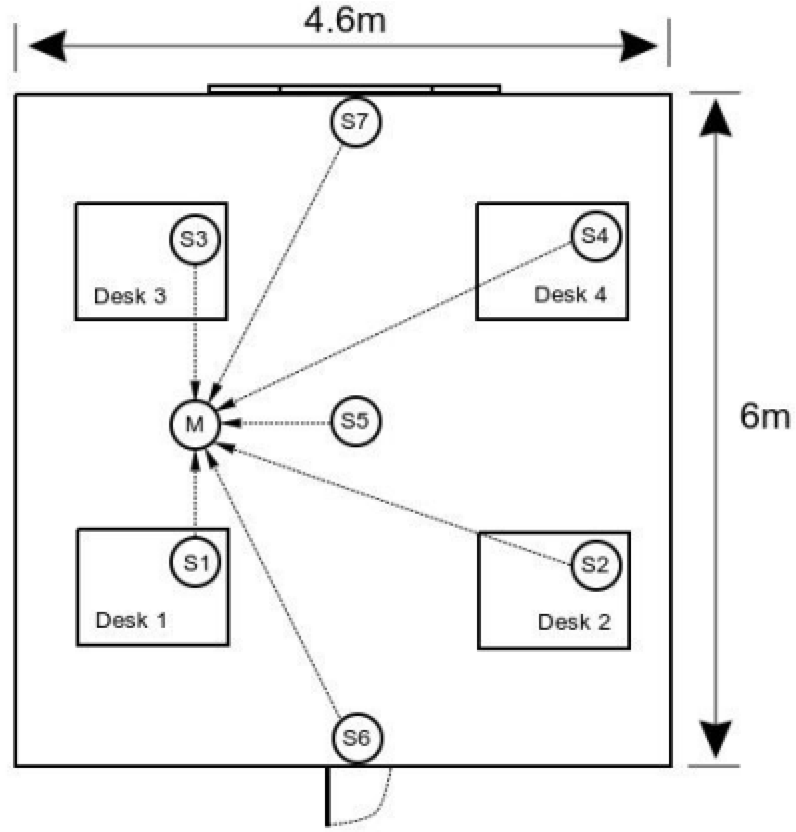
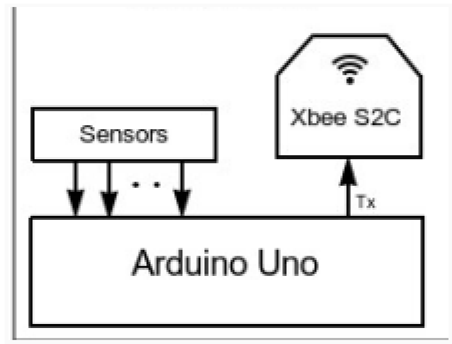


Fig. 1. A star network based data acquisition system deployed in a test room.

The experimental testbed for occupancy estimation is deployed in a 6m x 4.6m room. The setup consisted of 7 sensor nodes and one edge node in a star configuration, with the sensor nodes transmitting data to the edge every 30 seconds using wireless transceivers. Five different types of non-intrusive sensors were used in this experiment: temperature, light, sound, CO2, and digital passive infrared (PIR). The data was collected for four days in a controlled manner, with the occupancy in the room varying between 0 and 3 people. The ground truth of the occupancy count in the room was noted manually. Sensor nodes S1-S4 consisted of temperature, light and sound sensors, S5 had a CO2 sensor and S6 and S7 had one PIR sensor each that were deployed on the ceiling ledges at an angle that maximized the sensor's field of view for motion detection. Here sensor nodes S1-S4 each consist of 3 sensors whose block architecture is below.



The dataset consists of

- Dataset Characteristics- Multivariate, Time-Series
- Subject Area - Computer Science
- Associated Tasks - Classification
- Feature Type - Real
- # Instances - 10129
- # Features - 18

The attributes in our dataset are -

1. Date: YYYY/MM/DD
2. Time: HH:MM: SS
3. S1_Temp: In degrees Celsius
4. S2_Temp: In degrees Celsius
5. S3_Temp: In degrees Celsius
6. S4_Temp: In degrees Celsius
7. S1_Light: In Lux
8. S2_Light: In Lux
9. S3_Light: In Lux
10. S4_Light: In Lux
11. S1_Sound: In Volts (amplifier output read by ADC)

12. S2_Sound: In Volts (amplifier output read by ADC)
13. S3_Sound: In Volts (amplifier output read by ADC)
14. S4_Sound: In Volts (amplifier output read by ADC)
15. S5_CO2: In PPM0
16. S5_CO2 Slope: Slope of CO2 values taken in a sliding window
17. S6_PIR: Binary value conveying motion detection
18. S7_PIR: Binary value conveying motion detection

Data Analysis

First, we import all the necessary libraries required in our code:

Importing Libraries

```
[59] #Libraries for data manipulation
     import numpy as np
     import pandas as pd
     import re

     #Libraries for graph plotting
     import seaborn as sns
     import matplotlib.pyplot as plt

     #Libraries for ML Classification
     from sklearn import metrics
     import matplotlib.dates as mdates
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import confusion_matrix, accuracy_score, ConfusionMatrixDisplay
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import cross_val_score
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.preprocessing import LabelEncoder
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.svm import SVC
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import normalize
     from sklearn.preprocessing import StandardScaler
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import classification_report
     from sklearn.metrics import precision_score
     from sklearn.metrics import recall_score
     from sklearn.metrics import f1_score
```

Next, we read the dataset in the variable df, using pd.read_csv( ), and display its dimensions using the shape method:

```
[3]  PATH = "/content/Occupancy_Estimation.csv"
     df = pd.read_csv(PATH)

[5]  df.shape

     (10129, 22)
```

Next, we look at some standard data used for statistical analysis. Pandas has an inbuilt function for the same, and it displays the count, mean, standard deviation, minima, maxima, and quartile values for the attributes in our dataset.

```
[63] df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
S1_Temp	10129.0	25.454012	0.351351	24.940000	25.190000	25.38	25.63	26.380000
S2_Temp	10129.0	25.546059	0.586325	24.750000	25.190000	25.38	25.63	29.000000
S3_Temp	10129.0	25.056621	0.427283	24.440000	24.690000	24.94	25.38	26.190000
S4_Temp	10129.0	25.754125	0.356434	24.940000	25.440000	25.75	26.00	26.560000
S1_Light	10129.0	25.445059	51.011264	0.000000	0.000000	0.00	12.00	165.000000
S2_Light	10129.0	26.016290	67.304170	0.000000	0.000000	0.00	14.00	258.000000
S3_Light	10129.0	34.248494	58.400744	0.000000	0.000000	0.00	50.00	280.000000
S4_Light	10129.0	13.220259	19.602219	0.000000	0.000000	0.00	22.00	74.000000
S1_Sound	10129.0	0.168178	0.316709	0.060000	0.070000	0.08	0.08	3.880000
S2_Sound	10129.0	0.120066	0.266503	0.040000	0.050000	0.05	0.06	3.440000
S3_Sound	10129.0	0.158119	0.413637	0.040000	0.060000	0.06	0.07	3.670000
S4_Sound	10129.0	0.103840	0.120683	0.050000	0.060000	0.08	0.10	3.400000
S5_CO2	10129.0	460.860401	199.964940	345.000000	355.000000	360.00	465.00	1270.000000
S5_CO2_Slope	10129.0	-0.004830	1.164990	-6.296154	-0.046154	0.00	0.00	8.980769
S6_PIR	10129.0	0.090137	0.286392	0.000000	0.000000	0.00	0.00	1.000000
S7_PIR	10129.0	0.079574	0.270645	0.000000	0.000000	0.00	0.00	1.000000
Room_Occupancy_Count	10129.0	0.398559	0.893633	0.000000	0.000000	0.00	0.00	3.000000

Using *df.info( )*, we find the information about our dataset, i.e., how many attributes there are, the datatype of each attribute, and whether there is any null value to it or not. Here, we observe that our dataset has no null values and doesn't require any cleaning.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10129 entries, 0 to 10128
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  10129 non-null  object
 1   Time                  10129 non-null  object
 2   S1_Temp               10129 non-null  float64
 3   S2_Temp               10129 non-null  float64
 4   S3_Temp               10129 non-null  float64
 5   S4_Temp               10129 non-null  float64
 6   S1_Light              10129 non-null  int64
 7   S2_Light              10129 non-null  int64
 8   S3_Light              10129 non-null  int64
 9   S4_Light              10129 non-null  int64
 10  S1_Sound              10129 non-null  float64
 11  S2_Sound              10129 non-null  float64
 12  S3_Sound              10129 non-null  float64
 13  S4_Sound              10129 non-null  float64
 14  S5_CO2                10129 non-null  int64
 15  S5_CO2_Slope          10129 non-null  float64
 16  S6_PIR                10129 non-null  int64
 17  S7_PIR                10129 non-null  int64
 18  Room_Occupancy_Count  10129 non-null  int64
dtypes: float64(9), int64(8), object(2)
memory usage: 1.5+ MB
```

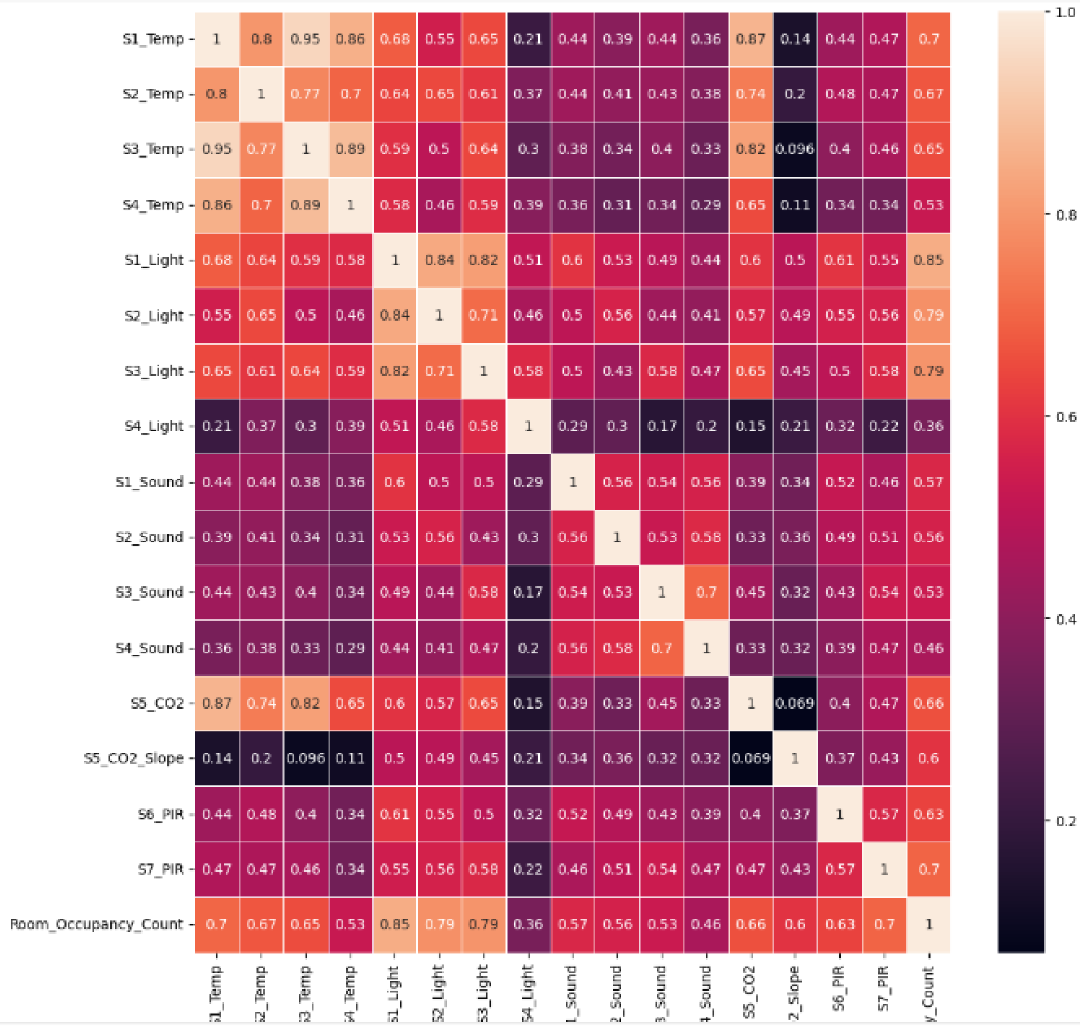
```
df.head()
```

	Date	Time	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Light	S1_Sound	S2_Sound	S3_Sound	S4_Sound	S5_CO2	S5_CO2_Slope	S6_PIR	S7_PIR	Room_Occupancy_Count
0	2017/12/22	10:49:41	24.94	24.75	24.56	25.38	121	34	53	40	0.08	0.19	0.06	0.06	390	0.769231	0	0	1
1	2017/12/22	10:50:12	24.94	24.75	24.56	25.44	121	33	53	40	0.93	0.05	0.06	0.06	390	0.646154	0	0	1
2	2017/12/22	10:50:42	25.00	24.75	24.50	25.44	121	34	53	40	0.43	0.11	0.08	0.06	390	0.519231	0	0	1
3	2017/12/22	10:51:13	25.00	24.75	24.56	25.44	121	34	53	40	0.41	0.10	0.10	0.09	390	0.388462	0	0	1
4	2017/12/22	10:51:44	25.00	24.75	24.56	25.44	121	34	54	40	0.18	0.06	0.06	0.06	390	0.253846	0	0	1

Here, we have used the *df.head()* function to display the first five entries of our dataset to demonstrate how our input dataset looks. Having a look at a dataset like this also makes it easier to work with and perform operations and make plots using it.

Heatmap of covariance between pairs of classes:

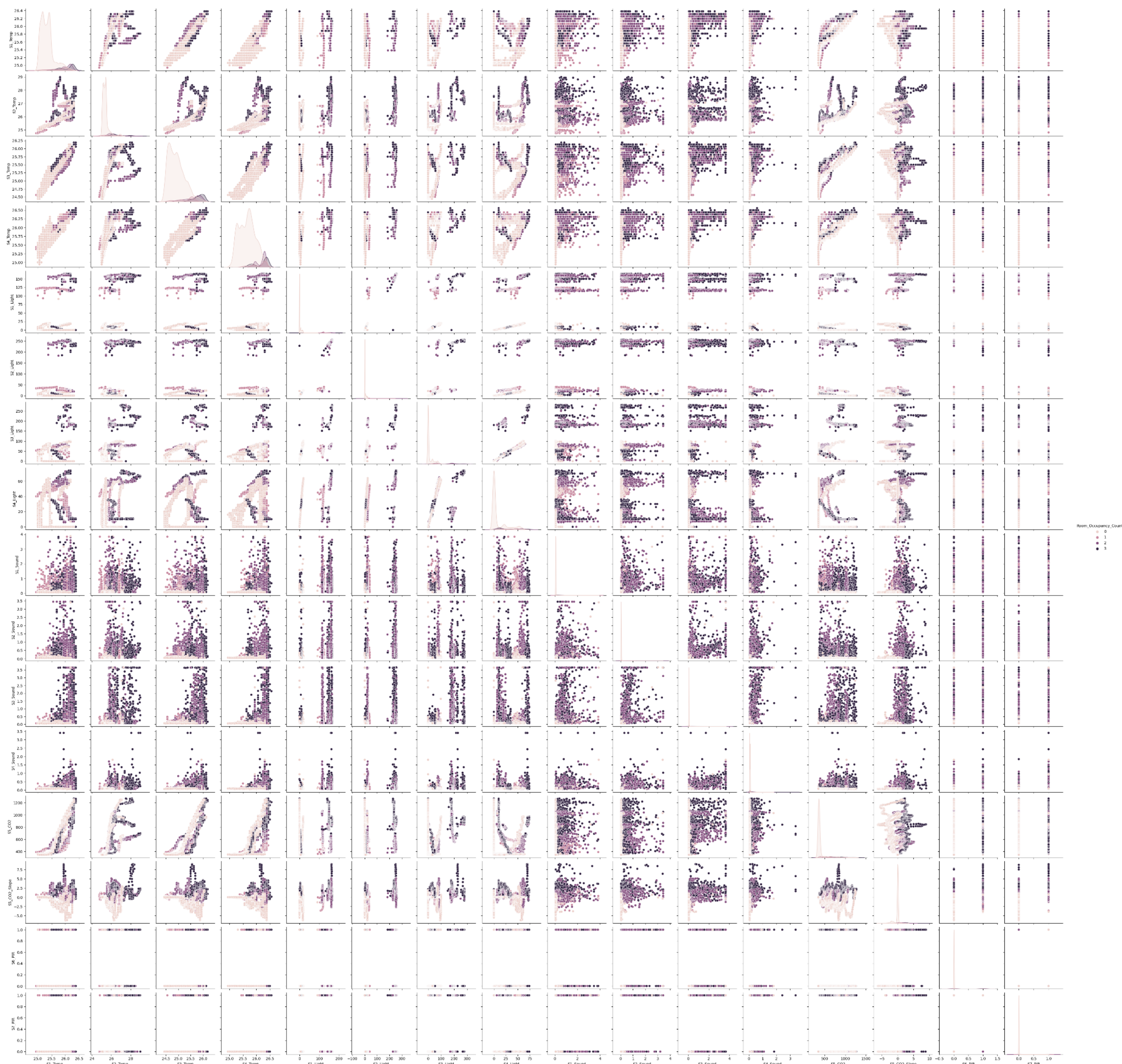
```python
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(12,12))
sns.heatmap(corr_matrix, annot=True, linewidth=.5, ax=ax)
```



All possible attribute pair plots:

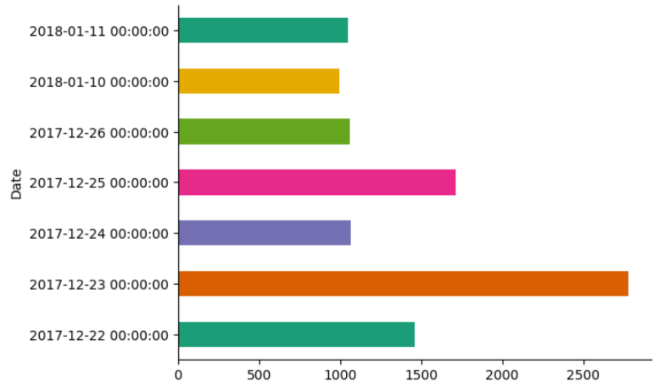
The different coloured points are a result of the “hue” attribute of the pair plot function which assigns a different colour to the 4 classes in the “Room_Occupancy_Count” attribute.

```python
sns.pairplot(df, hue = 'Room_Occupancy_Count')
plt.show()
```



Number of observations recorded on each day:

```
[ ] df.groupby('Date').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
```



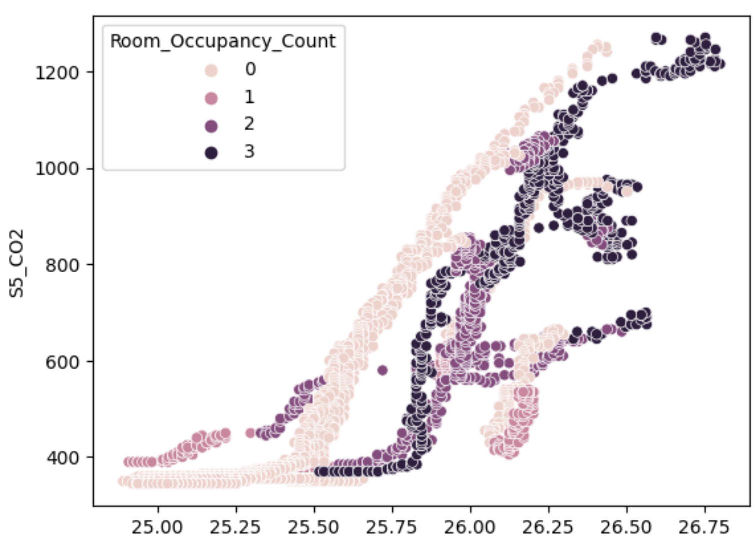
By grouping the dataset's rows by dates using the code in the image above, we can see that most observations were made on *23rd December 2017*, while the least number of observations were made on *10th January 2018*.

Pair Plot between the mean Temperature and S5_CO2 values:

Again, the different coloured points are a result of the "hue" attribute of the pair plot function, which assigns a different colour to the 4 classes in the "Room_Occupancy_Count" attribute.

The mean temperature values from S1_Temp, S2_Temp, S3_Temp, and S4_Temp are plotted against S5_CO2 values for easy visualisation.

```
1 sns.scatterplot(x=df.loc[: , "S1_Temp":"S4_Temp"].mean(axis=1), y=df["S5_CO2"], hue = df["Room_Occupancy_Count"])
```



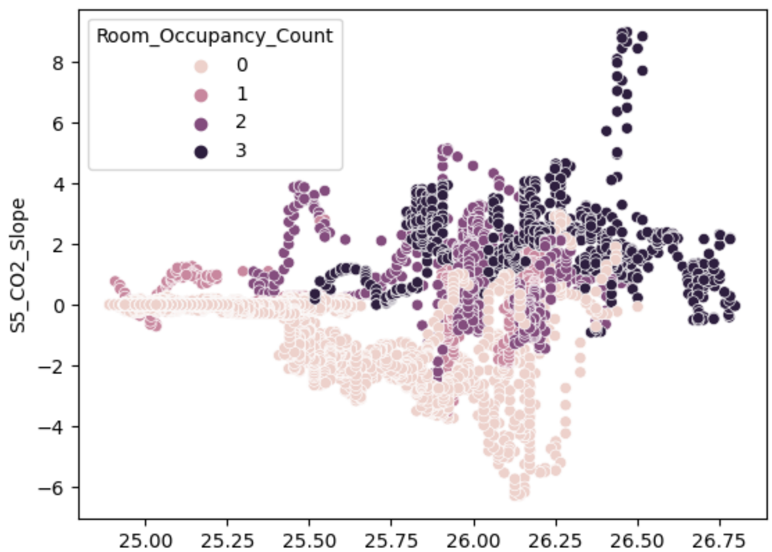
From this, we can make the following observations:

For a given occupancy level (Points of a single colour), the CO2 levels vs Temperature values roughly translate into a curve. This gives us the idea of utilising the value of the S5_CO2_Slope attribute of the dataset to derive some meaningful inferences.

Pair Plot between the mean Temperature and S5_CO2_Slope values:

Building on the previous inferences, we run the following code

```
sns.scatterplot(x=df.loc[: , "S1_Temp":"S4_Temp"].mean(axis=1), y=df["S5_CO2_Slope"], hue = df["Room_Occupancy_Count"])
```



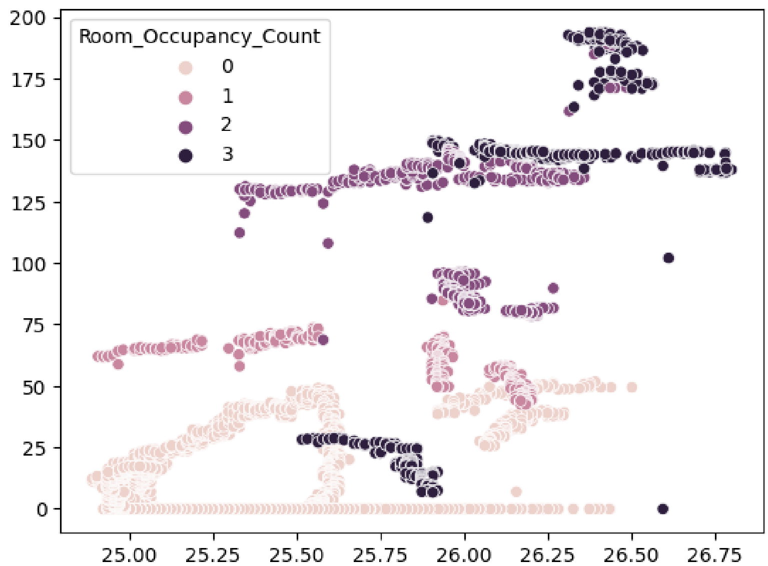
From the plot given above, we can derive some generalised inferences:

1. At a constant temperature, a higher level of the S5_CO2_Slope attribute indicates the presence of more people in the room.
2. Above a baseline S5_CO2_Slope value, a recorded temperature increase indicates more people in the room.

Pair plot between the mean Light values and mean Temperature values

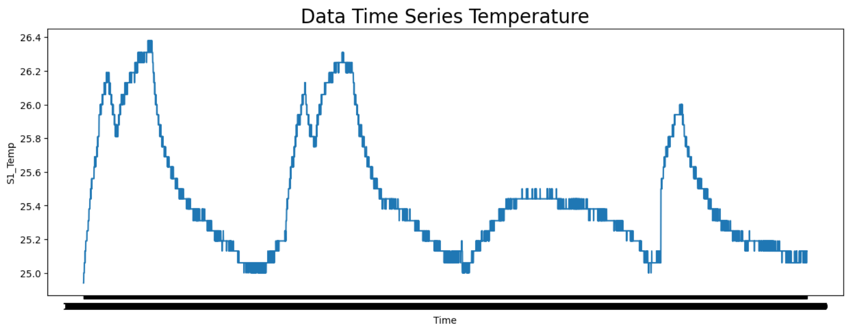
Like the previous plot, we use the mean values again to generate this plot with the following code

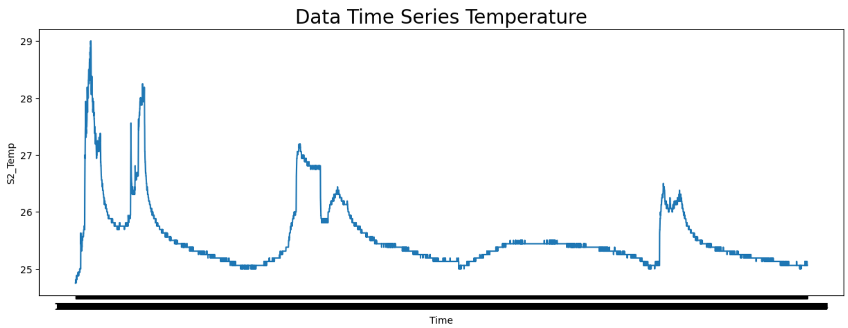
```
sns.scatterplot(x=df.loc[: , "S1_Temp":"S4_Temp"].mean(axis=1),
                y=df.loc[: , "S1_Light":"S4_Light"].mean(axis=1),
                hue = df["Room_Occupancy_Count"])
```

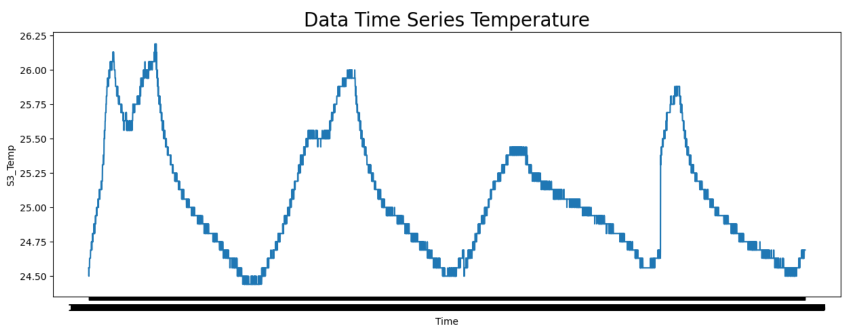


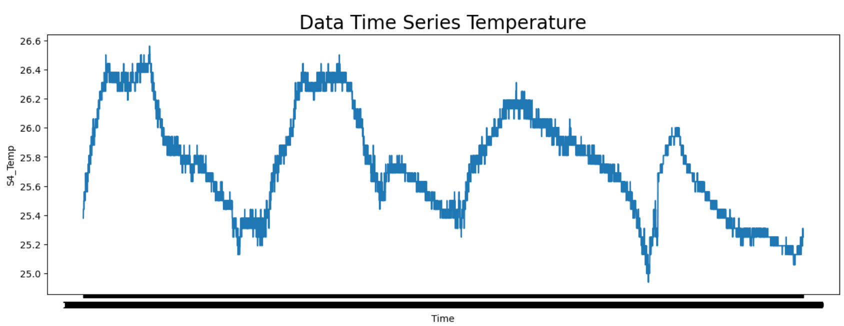
Here, too, we can derive an inference that for a constant temperature (On the x-axis), an increase in the Light Values (On the y-axis) generally indicates higher room occupancy, except for the approximate temperature range of 25.50 degrees Celsius and 25.9 degrees Celsius, where a differing trend is observed.

Time Series plots of Temperature for each sensor



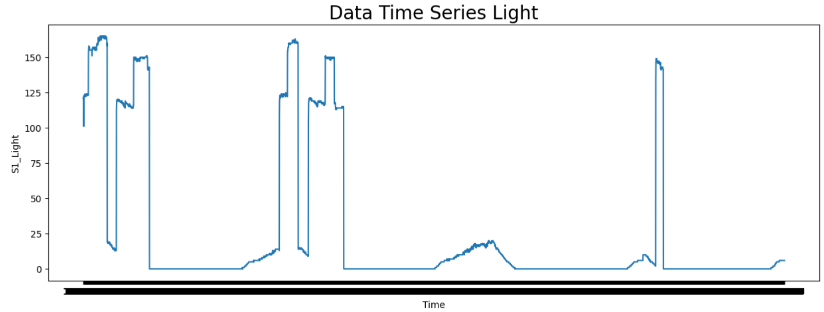


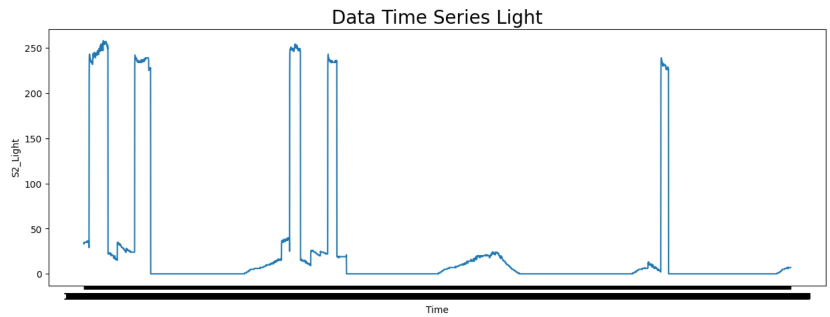


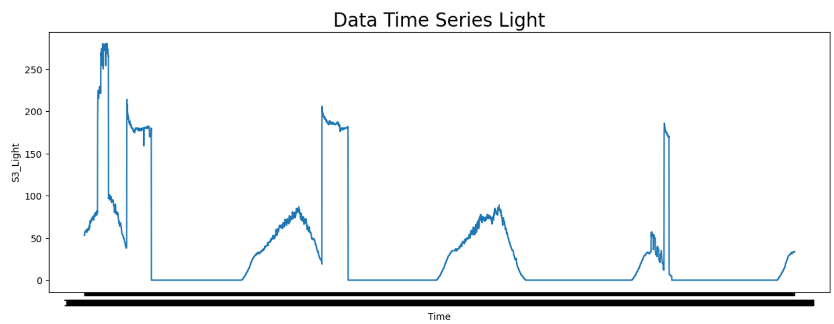


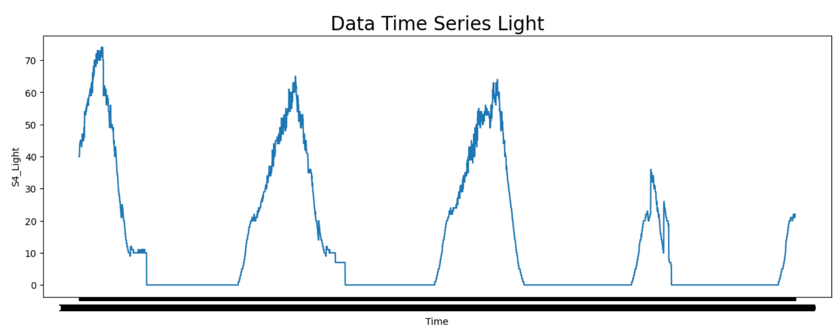
There is a reasonable similarity between the plots above (From S1 to S4), indicating that no major heat source/sink is present in the room or in the sensor's vicinity that may skew the temperature readings.

Time Series plots of Luminosity for each sensor







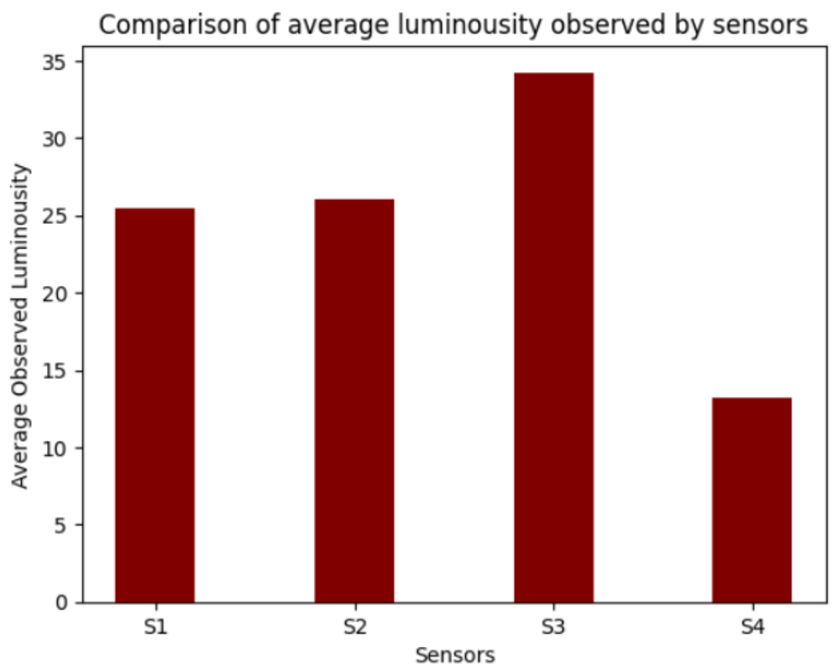


In the plots above (From S1 to S4) we see an evident change in the y-axis scale for the plot of S4. This indicates that S4 may be placed in a location with poor lighting, in comparison to the other sensors. Moreover, the values of S3 are also fairly higher than those of S1 and S2, indicating proximity to a light source, which leads us to look into this further.

Bar Chart of the mean luminosity values observed by sensors S1, S2, S3, and S4.

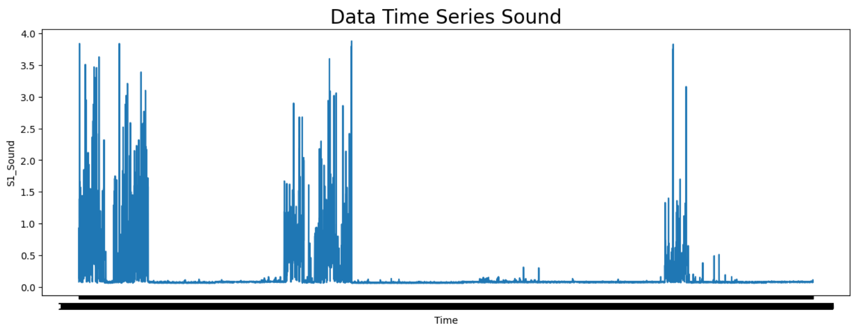
```
x = ["S1", "S2", "S3", "S4"]
y = [df.S1_Light.mean(), df.S2_Light.mean(), df.S3_Light.mean(), df.S4_Light.mean()]

plt.bar(x, y, color ='maroon',
        width = 0.4)

plt.xlabel("Sensors")
plt.ylabel("Average Observed Luminousity")
plt.title("Comparison of average luminousity observed by sensors")
```

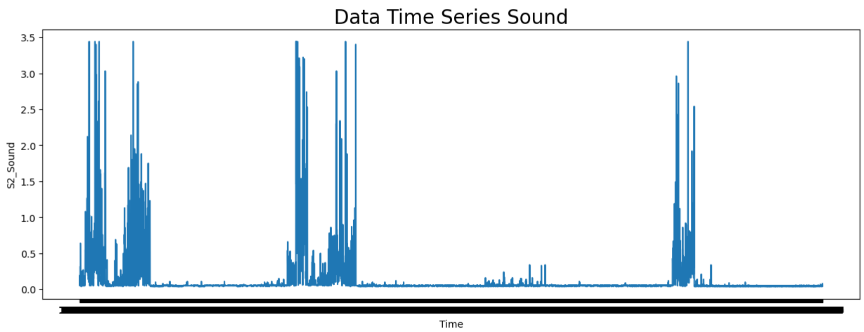
```
Text(0.5, 1.0, 'Comparison of average luminousity observed by sensors')
```

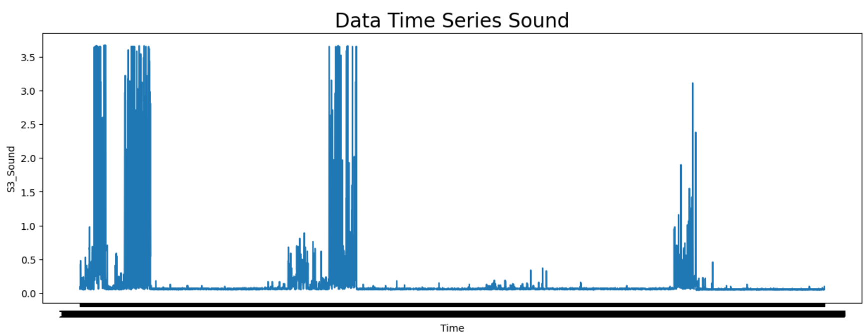


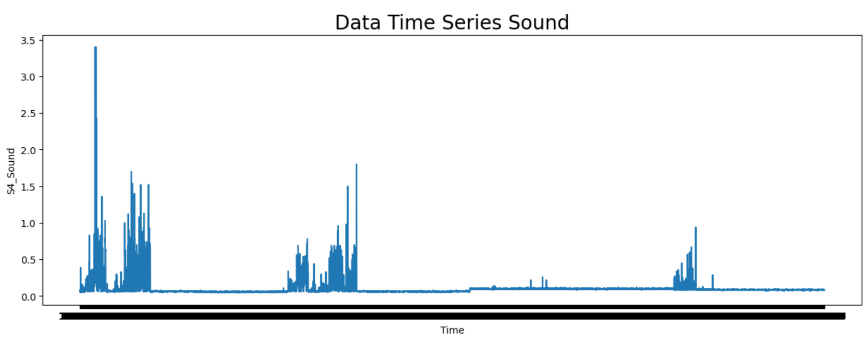
Here, the mean luminosity values observed by S1 and S2 are similar and lie almost exactly between the values observed by S3 and S4. Looking at this data and assuming all the sensors are working perfectly, we can infer that sensor S3 is probably placed closer to a light source, and S4 is placed away from a light source. This inference would also align with the above observations and the time series plots.

Time Series plots of Sound for each sensor







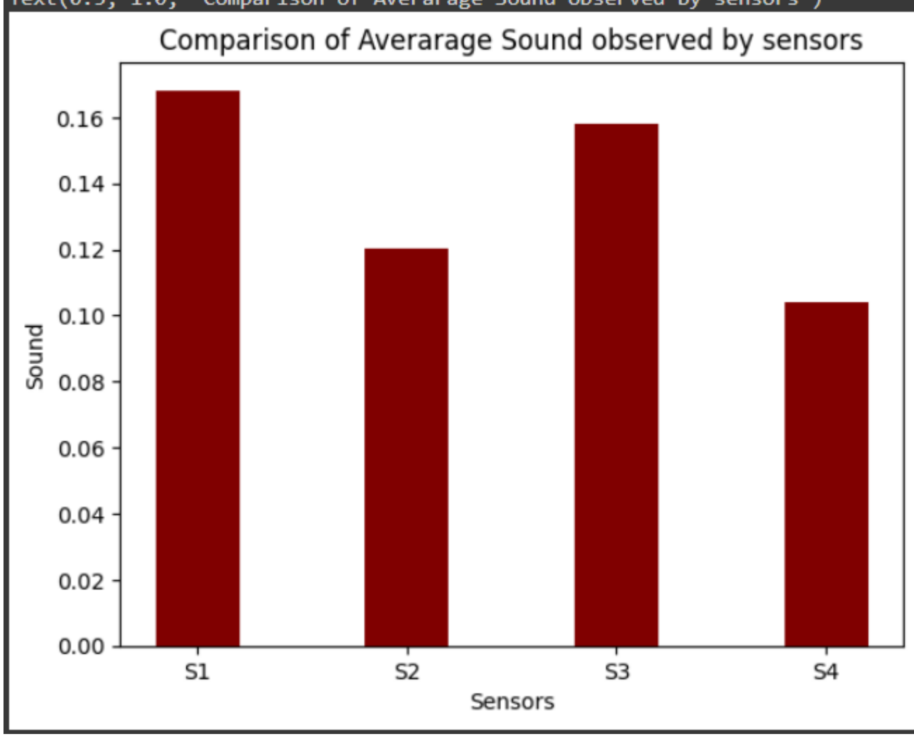


In the plots above (From S1 to S4), we see that S1 takes larger values than S2, S3 and S4, as evidenced by the scale of the plots. S4 also takes the least values. Indicating that S1 may be in a position where more people frequent the room, and likewise, the position of S4 is in a location people don’t frequent often, hence giving us the lower values.

Bar Chart of the mean sound values observed by sensors S1, S2, S3, and S4.

```
x = ["S1", "S2", "S3", "S4"]
y = [df.S1_Sound.mean(), df.S2_Sound.mean(), df.S3_Sound.mean(), df.S4_Sound.mean()]

plt.bar(x, y, color ='maroon',
        width = 0.4)

plt.xlabel("Sensors")
plt.ylabel("Sound")
plt.title("Comparison of Average Sound observed by sensors")
```

```
Text(0.5, 1.0, 'Comparison of Averarage Sound observed by sensors')
```



Here, the mean sound values are consistent with the predictions above, with the highest importance for S1 and the lowest for S4. Consequently indicating that the above made the inferences have a strong basis to be true. S1 is near the locations people frequent, hence capturing higher readings, and S4 is a region least frequented by people, resulting in lower readings being captured, on average.

ML Classification Algorithms

After thoroughly analysing and updating the dataset, we apply a machine learning model to predict the occupancy count. Instead of applying just one, our project uses 5 different classification algorithms to predict the results and compare the results between them to see which algorithm works best for our given dataset.

We applied the algorithms given below:

1. Logistic regression
2. Decision Tree Classifier
3. Naive Bayes Classifier
4. Random Forest Classifier
5. Support Vector Machine

Using the below 80/20 train-test split:

```
X = df.drop(['Room_Occupancy_Count'], axis=1)
y = df.loc[:,['Room_Occupancy_Count']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 26)
print("Shape of X_train: ",X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ",y_train.shape)
print("Shape of y_test",y_test.shape)
```

```
Shape of X_train:  (8103, 19)
Shape of X_test:  (2026, 19)
Shape of y_train:  (8103, 1)
Shape of y_test (2026, 1)
```

Logistic Regression

Multinomial logistic regression is a direct extension of logistic regression to handle multi-class problems without employing binary classification strategies like using Sigmoid Function.

Working Principle:
It models the probabilities of multiple classes directly, using a single model with multiple output classes.

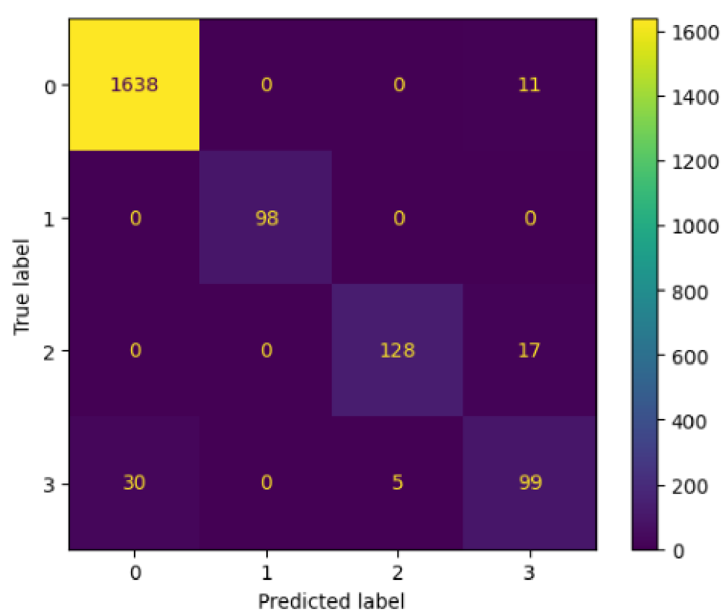
Instead of a binary outcome, it predicts probabilities for each class using the softmax function, ensuring that the sum of probabilities across all classes equals one.

Our Implementation:

```
model1 = LogisticRegression()
model1.fit(X_train, y_train)
y_pred = model1.predict(X_test)
```

```
[31] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))

     Accuracy Score = 96.89042448173741
```

```
[32] cm = confusion_matrix(y_test, y_pred, labels=model1.classes_)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                   display_labels=model1.classes_)
     disp.plot()
     plt.show()
```



```
print(classification_report(y_test,y_pred,digits=3))
```

```
              precision    recall  f1-score   support

           0      0.982     0.993     0.988      1649
           1      1.000     1.000     1.000        98
           2      0.962     0.883     0.921       145
           3      0.780     0.739     0.759       134

    accuracy                          0.969      2026
   macro avg      0.931     0.904     0.917      2026
weighted avg      0.968     0.969     0.968      2026
```

Accuracy: 96.89%
This model has lesser accuracy than most of the implementations in our report, but we still used this as it’s one of the elementary classification implementations. We also see that the F-Score for class 3 is very low when compared to the other classes. Hence making it a less than ideal model.

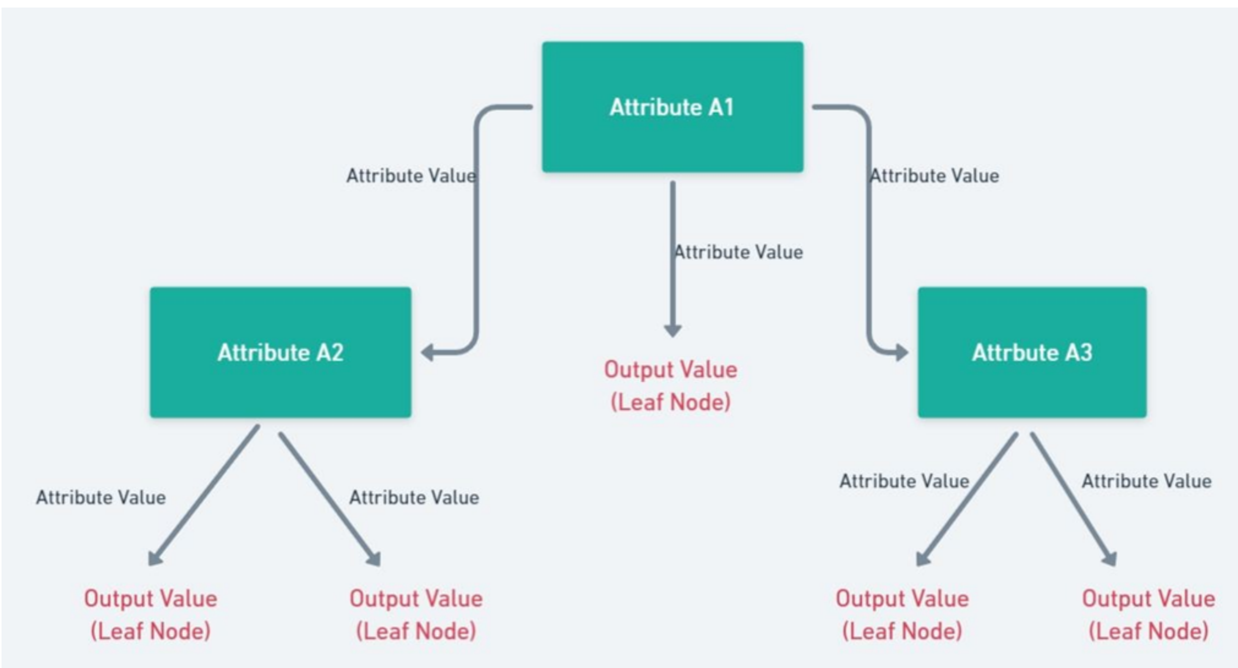
Decision Tree Classifier

Trees are hierarchical tree-like structures used for classification and regression tasks. They partition the dataset into smaller subsets based on feature values, aiming to create homogeneous leaf nodes.

Working Principle:

- **Tree Construction:** Decision Trees start at the root node and recursively split the data based on feature values to maximize information gain (for classification) or minimize impurity (for regression). Each split divides the data into branches. leading to nodes until reaching leaf nodes where predictions are made.

- **Feature Selection:** The algorithm selects the best split at each node based on criteria like Gini impurity, entropy, or information gain. It continues splitting until a stopping criterion (like a maximum depth or minimum number of samples per leaf) is reached.



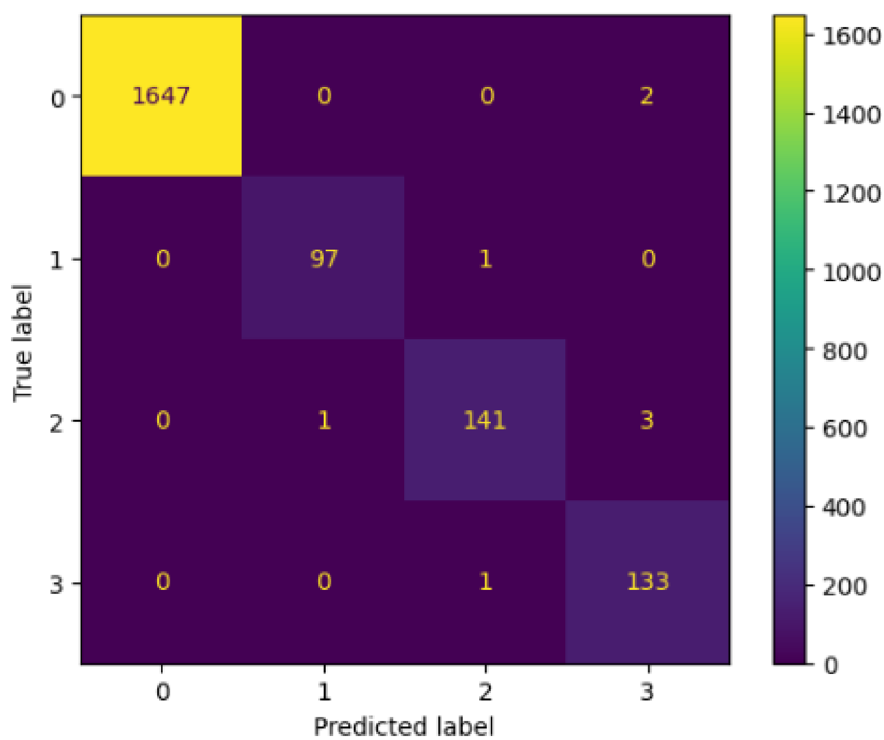
Our Implementation:

```
[50] model2 = DecisionTreeClassifier()
     model2.fit(X_train, y_train)
     y_pred = model2.predict(X_test)
```

```
[51] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))
```

```
Accuracy Score = 99.60513326752222
```

```
[36] cm = confusion_matrix(y_test, y_pred, labels=model2.classes_)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                   display_labels=model2.classes_)
     disp.plot()
     plt.show()
```



```
print(classification_report(y_test,y_pred,digits=3))
```

```
              precision    recall  f1-score   support

           0      1.000     0.999     0.999      1649
           1      0.990     0.990     0.990        98
           2      0.986     0.972     0.979       145
           3      0.964     0.993     0.978       134

    accuracy                          0.996      2026
   macro avg      0.985     0.988     0.987      2026
weighted avg      0.996     0.996     0.996      2026
```

Accuracy: 99.605%
This model gives us a very solid and consistent performance. Moreover, the F-values obtained in this implementation are also acceptable. This intuitively also tells us that models with similar approaches should also fare well to solve our problem statement.

Gaussian Naive Bayes Classifier

Gaussian Naive Bayes assumes that continuous features follow a Gaussian distribution.

Working Principle:

- It calculates the likelihood of a particular feature value given the class by assuming a Gaussian (normal) distribution for each class-feature combination.
- It estimates the mean and variance for each class-feature pair from the training data and uses them to compute probabilities.

x represents features calculated individually.

Likelihood   Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

Posterior Probability   Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Where,

- $P(c|x)$ is the posterior probability of *class* c given *predictor* (*features*).
- $P(c)$ is the probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

And if we take the below assumption, then it becomes a Naive Bayes Classifier.

- Assume independence among attributes X_i when class is given:
  - $P(X_1, X_2, \ldots, X_d | Y_j) = P(X_1 | Y_j)\, P(X_2 | Y_j) \ldots P(X_d | Y_j)$

Our Implementation:

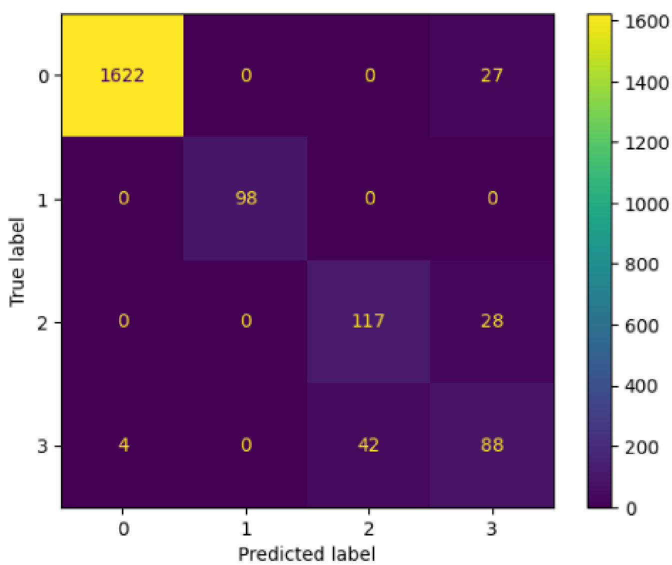
```
model3 = GaussianNB()
model3.fit(X_train, y_train)
y_pred = model3.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y wa
  y = column_or_1d(y, warn=True)
```

```
[39] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))
```

```
Accuracy Score = 95.01480750246792
```

```
[40] cm = confusion_matrix(y_test, y_pred, labels=model3.classes_)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                   display_labels=model3.classes_)
     disp.plot()
     plt.show()
```




```
[41] print(classification_report(y_test,y_pred,digits=3))
```

```
              precision    recall  f1-score   support

           0      0.998     0.984     0.991      1649
           1      1.000     1.000     1.000        98
           2      0.736     0.807     0.770       145
           3      0.615     0.657     0.635       134

    accuracy                          0.950      2026
   macro avg      0.837     0.862     0.849      2026
weighted avg      0.954     0.950     0.952      2026
```

Accuracy: 95.01%

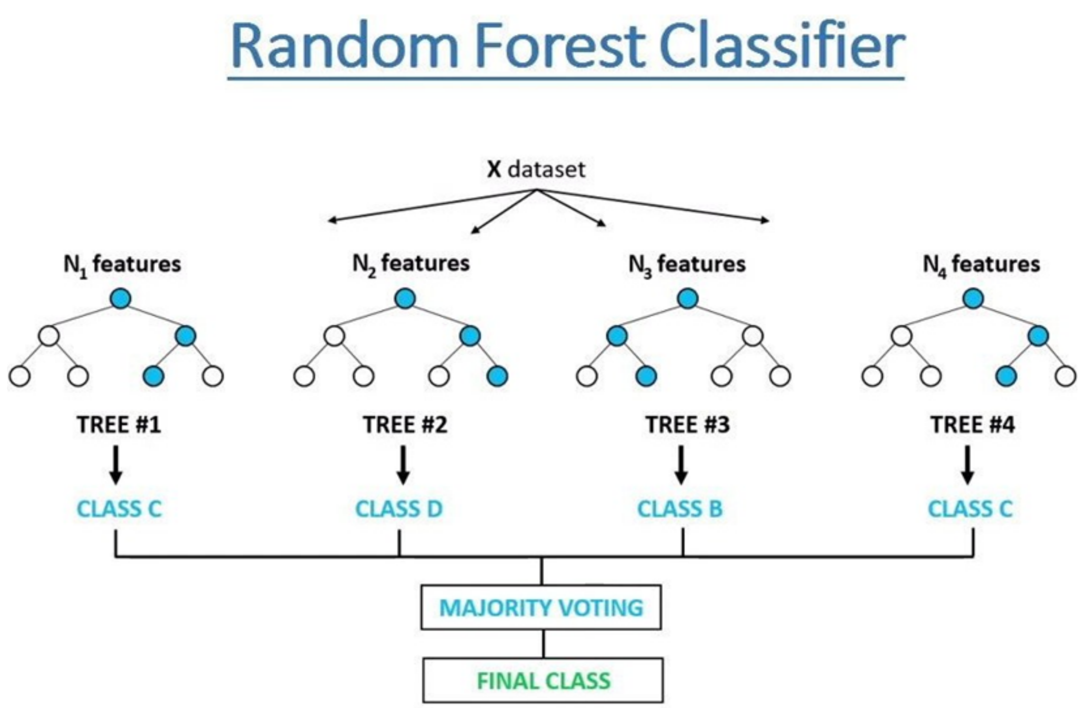
This implementation gives us the worst accuracy among all the models we've tested. Moreover, the obtained F-Values for each class aren't satisfactory either.

Random Forest Classifier

Random forest is a commonly used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result for improving accuracy and reducing overfitting.

Working Principle:

- **Multiple Trees:** Random Forest builds a collection of Decision Trees by bootstrapping the dataset (sampling with replacement) and selecting a subset of features at each node.
- **Aggregation:** Each tree in the forest independently makes predictions, and the final prediction is determined by averaging (for regression) or voting (for classification) across all trees.
- **Reducing Overfitting:** Using random subsets of data and features for each tree, Random Forest introduces randomness, leading to diverse trees. The ensemble nature helps in generalizing new data well.



Working of Random Forest

Our Implementation:

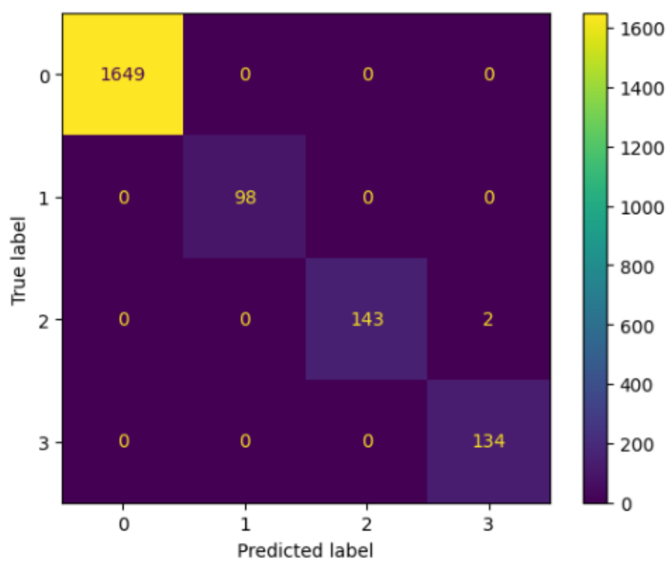
```
[42] model4 = RandomForestClassifier()
     model4.fit(X_train, y_train)
     y_pred = model4.predict(X_test)
```

```
<ipython-input-42-30621d211b3c>:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected.
  model4.fit(X_train, y_train)
```

```
[43] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))
```

```
Accuracy Score = 99.90128331688055
```

```
[44] cm = confusion_matrix(y_test, y_pred, labels=model4.classes_)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                   display_labels=model4.classes_)
     disp.plot()
     plt.show()
```




```
[45] print(classification_report(y_test,y_pred,digits=3))
```

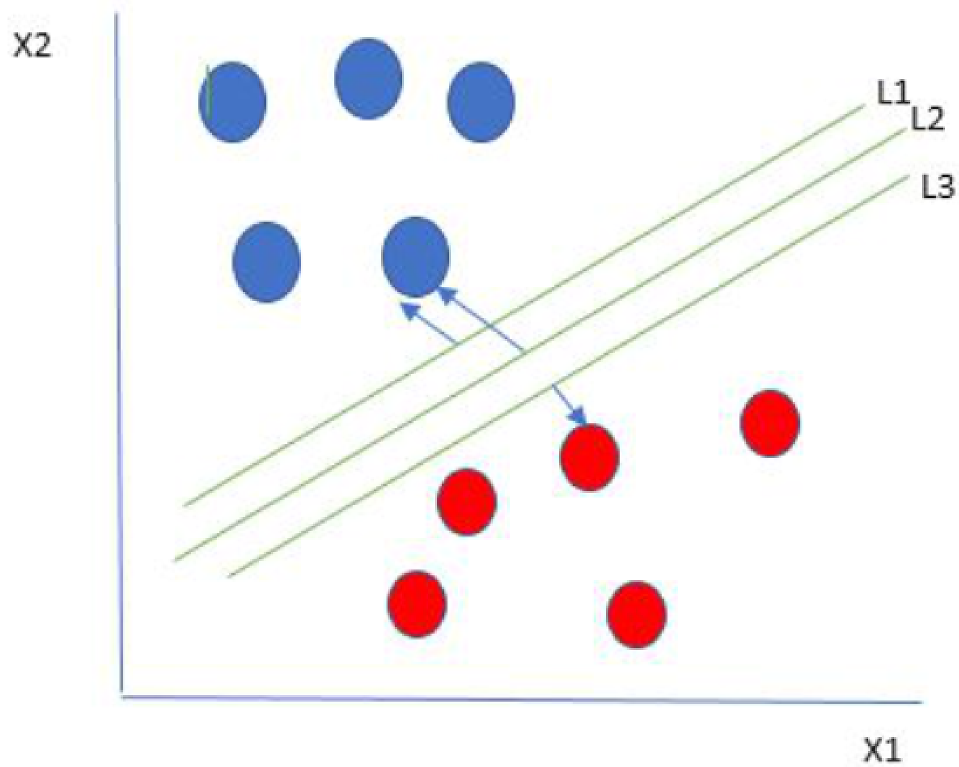
```
              precision    recall  f1-score   support

           0      1.000     1.000     1.000      1649
           1      1.000     1.000     1.000        98
           2      1.000     0.986     0.993       145
           3      0.985     1.000     0.993       134

    accuracy                          0.999      2026
   macro avg      0.996     0.997     0.996      2026
weighted avg      0.999     0.999     0.999      2026
```

Accuracy: 99.90%

This implementation using the random forest model seems to give us the best model for the given problem. It also has the highest F-Values for each class among all the models we have tested.

Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features.



Multiple hyperplanes separate the data from two classes

Our Implementation:

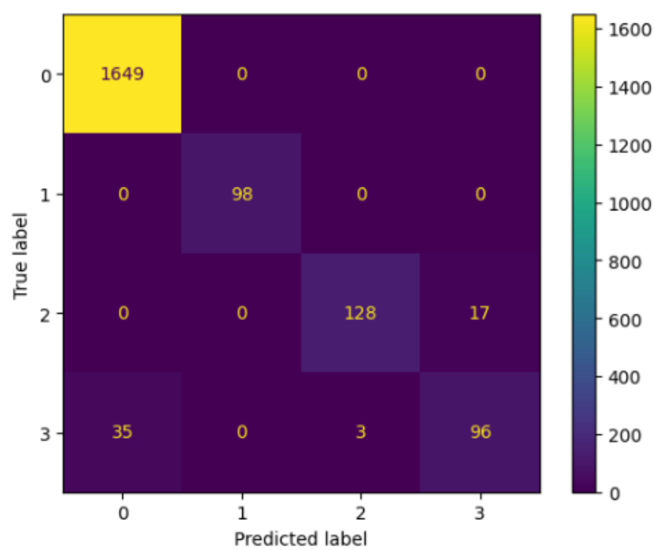
```
[46] model5 = SVC()
     model5.fit(X_train, y_train)
     y_pred = model5.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y wa
  y = column_or_1d(y, warn=True)
```

```
[47] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))
```

```
Accuracy Score = 97.28529121421519
```

```
[48] cm = confusion_matrix(y_test, y_pred, labels=model5.classes_)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                   display_labels=model5.classes_)
     disp.plot()
     plt.show()
```




```
print(classification_report(y_test,y_pred,digits=3))
```

```
              precision    recall  f1-score   support

           0      0.979     1.000     0.989      1649
           1      1.000     1.000     1.000        98
           2      0.977     0.883     0.928       145
           3      0.850     0.716     0.777       134

    accuracy                          0.973      2026
   macro avg      0.951     0.900     0.924      2026
weighted avg      0.971     0.973     0.972      2026
```

Accuracy: 97.285%

While this implementation has decent accuracy, we see that the F-Values for class 3 could be better.

Conclusion

To summarize our results from the implementations, we get the following table:

Algorithm	**Accuracy**
Logistic Regression	96.89%
Decision Tree	99.605%
Naive Bayes	95.01%
Random Forest	99.90%
Support Vector Machine	97.285%

From the results, it's clear that the Random Forest Classifier gives the highest accuracy of 99.90% on our dataset. Moreover, we can come to this intuitively as well since most of our inferences in the Data Analysis section involved multiple conditions by using at least 3 or more classes simultaneously.

We've uploaded the entire code on GitHub as well. Link to the code - https://github.com/RonitGupta2002/IDS_Project

References

- Course material for IDS.
- Official documentation for Seaborn, Matplotlib, sci-kit-learn, Pandas, and Numpy.
- Introductory Paper for this Dataset: https://archive.ics.uci.edu/dataset/864/room+occupancy+estimation
- Singh, Adarsh Pal et al. “Machine Learning-Based Occupancy Estimation Using Multivariate Sensor Nodes.” 2018 IEEE Globecom Workshops (GC Wkshps) (2018): 1-6.