Int.to Artificial Intelligence

UCK 358E: HOMEWORK 1
CRN:21505

Assoc. Prof. Barış Başpınar

Egehan Korkmaz* - **110200148**

*

Library Imports

To implement and evaluate the machine learning models, several Python libraries were imported. These include:

- `pandas` and `numpy` for data manipulation,
- `matplotlib.pyplot` and `seaborn` for data visualization,
- `scikit-learn` modules for preprocessing, model training, and evaluation.

The necessary classifiers (e.g., `LogisticRegression`, `DecisionTreeClassifier`) and tools such as `StandardScaler`, `accuracy_score`, and `confusion_matrix` were specifically imported to support training, testing, and performance evaluation steps of the occupancy detection models.

Data Loading and Exploration

The training dataset (`datatraining.txt`) was loaded using `pandas`. The dataset contains 8143 rows and 7 columns, including environmental features such as Temperature, Humidity, Light, CO_2, and HumidityRatio, along with the corresponding `Occupancy` label. An initial inspection using `df.head()` provided insight into the structure and value ranges of the data, confirming that it is suitable for further preprocessing and model development.

Initial Data Exploration

To better understand the structure and quality of the dataset, the `df.info()` and `df.describe()` functions were used. The dataset contains 8143 complete records with no missing values, covering 7 features. The data types include one object-type column (date), one integer (Occupancy), and five floating-point features (Temperature, Humidity, Light, CO_2, HumidityRatio). Summary statistics revealed considerable variability, especially in `Light` and `CO`$_2$ values, suggesting their potential importance in detecting occupancy. The occupancy rate was approximately 21%, indicating a class imbalance that must be considered during model evaluation.

Duplicate Data Check

To ensure the quality and uniqueness of the dataset, the presence of duplicate records was evaluated using the `df.duplicated()` function. The result indicated that there were no duplicate entries in the dataset. This confirms the reliability of the data and eliminates the need for any row-level filtering prior to model training.

Correlation Analysis

To understand the relationships between variables, a correlation heatmap was generated using Pearson's correlation coefficient. The correlation matrix shows that `Light` and `CO`$_2$ are highly correlated with the target variable `Occupancy`, with correlation coefficients of 0.91 and 0.71 respectively. Although the correlation of `Temperature` with occupancy is relatively lower (0.54), it is still considered informative and useful as a feature.
The analysis also revealed a very strong correlation (0.96) between `Humidity` and `HumidityRatio`, which indicates multicollinearity. High correlation between input features can cause redundancy and may negatively impact model performance, especially for linear models, by inflating the variance of estimated coefficients. Therefore, only one of these features should be selected. Given that `HumidityRatio` has a higher correlation with `Occupancy` (0.30) compared to `Humidity` (0.13), it was preferred during feature selection.

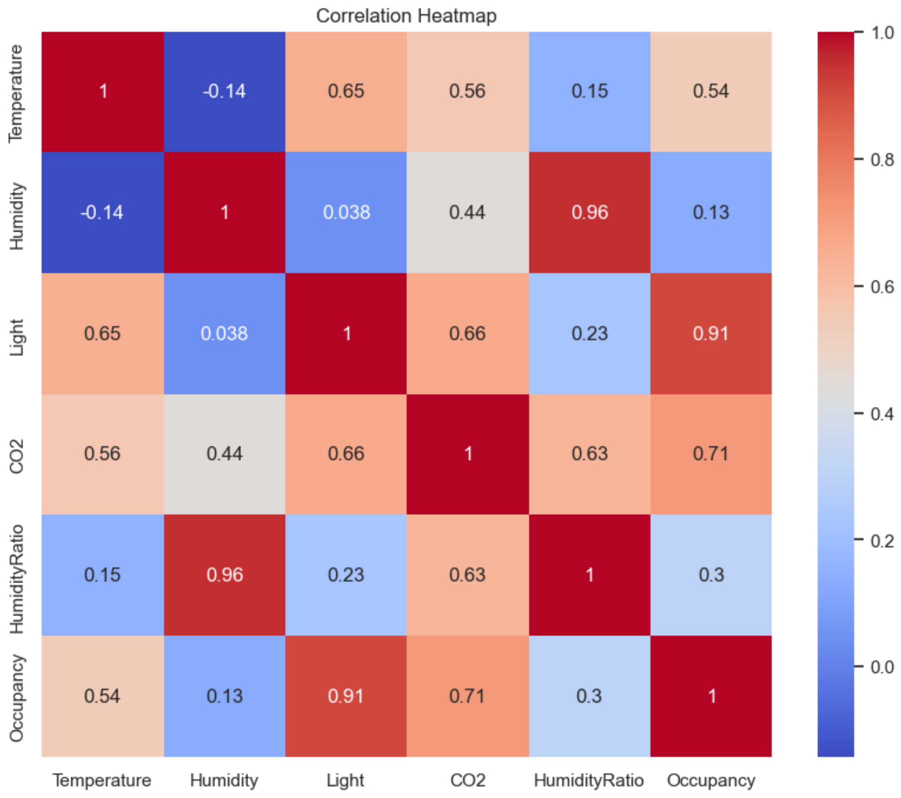


Figure 1: Correlation heatmap of the training dataset.

Feature Distributions by Occupancy

To examine how environmental variables differ between occupied and unoccupied states, distribution plots were generated for each feature, separated by the `Occupancy` class. The plots clearly show that some features, such as `Light` and `CO`$_2$, exhibit significant differences between the two occupancy states.

Light: This variable stands out as the most discriminative feature. Occupied rooms (`Occupancy=1`) show a strong concentration of values in the range of approximately 430 to 500 lux, whereas unoccupied rooms are heavily clustered below 200 lux. This stark contrast suggests that light levels are a powerful indicator of room usage.
CO_2: Similarly, `CO`$_2$ levels above approximately 750 ppm are associated with occupancy. This aligns with expected human metabolic output and makes CO_2 a biologically plausible predictor.
Temperature: A moderate shift is observed in temperature values, with occupancy generally associated with temperatures between 20.5°C and 22.5°C. However, since the overall range of temperatures in the dataset is already narrow (19–23°C), its predictive power is comparatively limited.
Humidity and HumidityRatio: These two features do not provide a clearly separable pattern based on the distribution plots. While their correlation with each other is extremely high (0.96), the class-wise distribution overlaps substantially. This overlap may be due to these variables being more sensitive to ambient conditions rather than occupancy status directly. Despite this, `HumidityRatio` shows slightly better separation and was therefore preferred during feature selection.

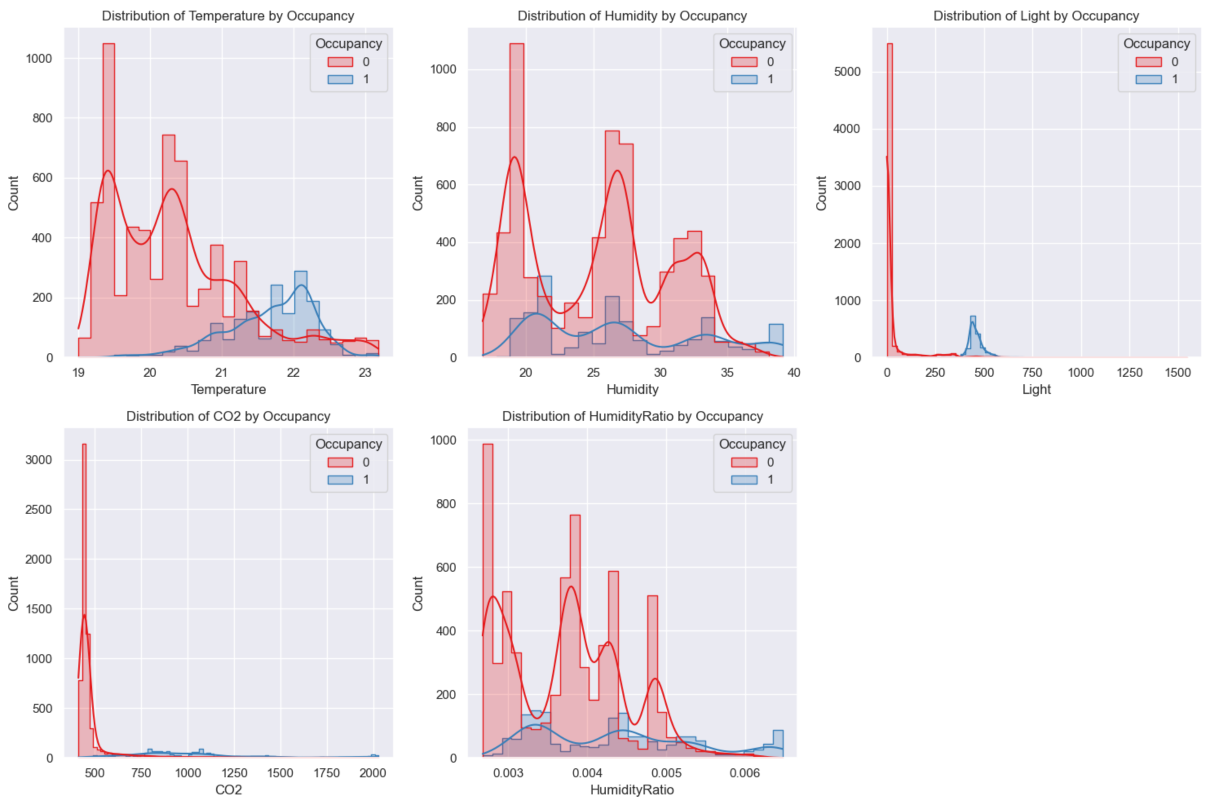


Figure 2: Distribution of features by Occupancy (0: unoccupied, 1: occupied).

Pairplot Analysis

To visualize the relationships between numerical features and their separability with respect to the `Occupancy` class, a pairplot was generated using Seaborn. This matrix of scatter plots and kernel density estimates provides valuable insights into feature interactions and class clustering behavior.
From the pairwise scatter plots, several patterns emerge:

- **Light vs. CO_2** and **Light vs. Temperature** combinations exhibit clear separation between occupied and unoccupied instances, validating previous findings from the correlation and distribution plots.
- **Light** once again emerges as the most powerful discriminating feature. High light intensity (typically in the range of 430–500 lux) is consistently associated with occupied states.
- **CO_2** values above approximately 750 ppm show a strong association with `Occupancy = 1`, indicating its effectiveness as a biological indicator of human presence.
- A moderate clustering of `Occupancy = 1` is observed when `Humidity` is between 30 and 35, and `HumidityRatio` lies between 0.005 and 0.0055. However, the separation is not as strong or as visually distinct as with `Light` and `CO`$_2$.

These patterns support the earlier findings from the distribution plots and correlation heatmap, reinforcing the significance of these variables in the classification task. To verify the previously observed clustering pattern, a conditional filter was applied to select data points where `Humidity` ranges between 30 and 35, and `HumidityRatio` falls between 0.005 and 0.0055. The following Python code was used for this filtering:

```python
filtered = df[(df["Humidity"] >= 30) & (df["Humidity"] <= 35) &
                 (df["HumidityRatio"] >= 0.005) & (df["HumidityRatio"
                   ] <= 0.0055)]
print(filtered["Occupancy"].value_counts())
```

Listing 1: Filtering data by Humidity and HumidityRatio range

The result indicated that there were 218 occupied instances and 133 unoccupied ones within this range. Although this suggests a slightly higher number of occupied samples, the difference is not strong enough to imply a statistically significant relationship. Therefore, we conclude that these features alone are not reliable predictors and should be used in combination with stronger indicators like `Light` and `CO`$_2$.

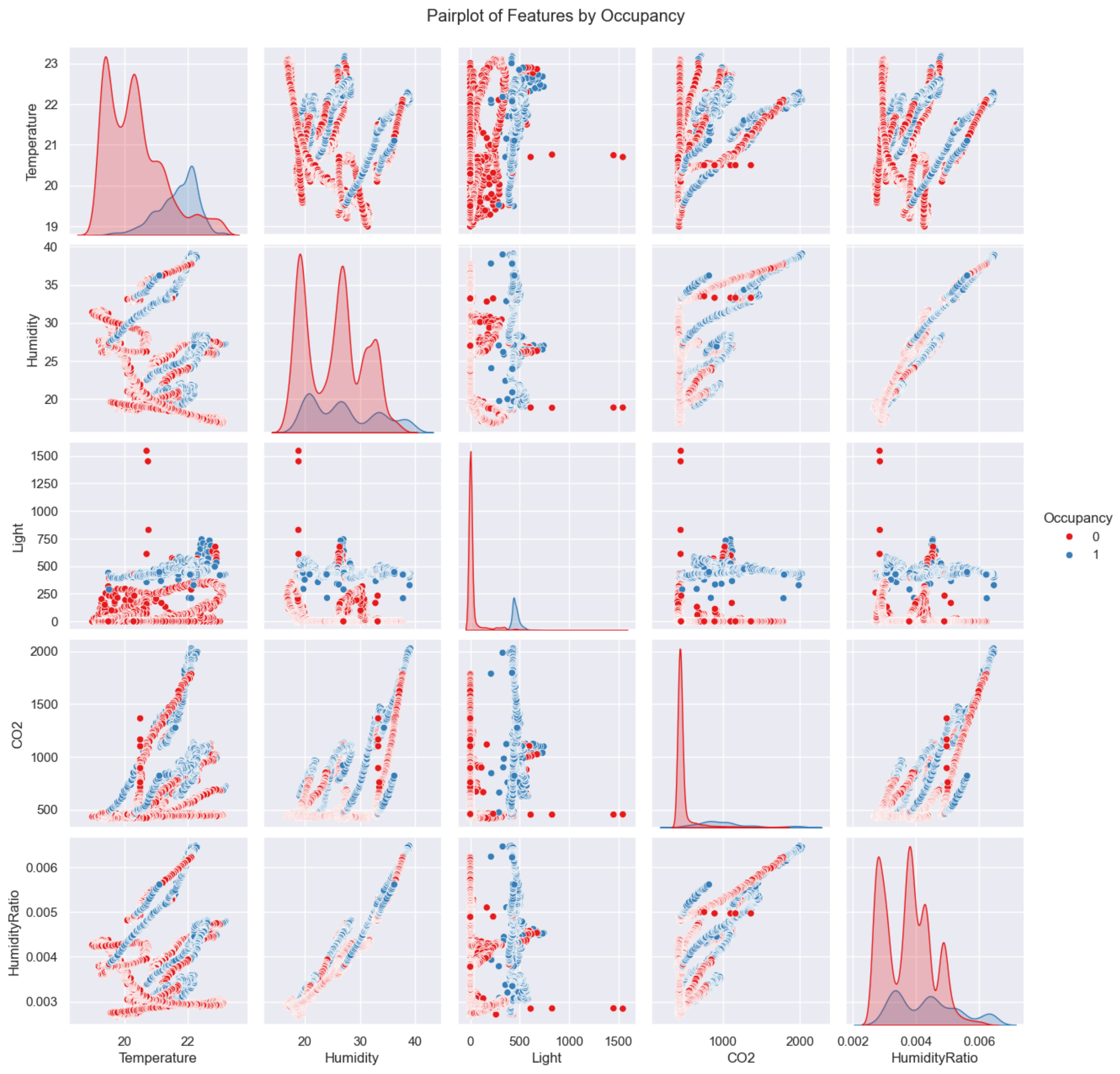


Figure 3: Pairplot of numerical features colored by Occupancy.

To verify the previously observed clustering pattern, a conditional filter was applied to select data points where `Humidity` ranges between 30 and 35, and `HumidityRatio` falls between 0.005 and 0.0055. Within this subset, 218 instances were found to be occupied (`Occupancy=1`) and 133 were unoccupied. Although there appears to be a slightly higher number of occupied instances in this specific range, the imbalance is not statistically significant enough to conclude a strong predictive relationship. Therefore, we consider that these features alone do not effectively predict occupancy and should be used in combination with stronger indicators like `Light` and `CO`$_2$.

Boxplot Analysis

To further investigate the distribution and separation capacity of each feature with respect to occupancy status, boxplots were generated for all five numerical variables. These visualizations allow direct comparison of the feature distributions between `Occupancy = 0` (unoccupied) and `Occupancy = 1` (occupied) groups, highlighting both the central tendency and spread.

- **Light** stands out as the most discriminative feature. Its median and interquartile range do not overlap between the two classes, and values under occupied conditions are notably higher and more consistent, mostly between 430–500 lux.
- **CO_2** also exhibits a significant shift between classes. Occupied states show a much higher and more variable CO_2 concentration (often exceeding 750 ppm), while unoccupied cases remain in a narrow lower range.
- **Temperature**, **Humidity**, and **HumidityRatio** show only moderate shifts in their distributions. While their medians shift slightly toward higher values under occupancy, the overlapping interquartile ranges indicate limited discriminative power when used individually.

These findings confirm and strengthen previous insights from the correlation heatmap, distribution plots, and pairplots: Light and CO_2 are the most relevant features for predicting occupancy, while other features are better used in combination.

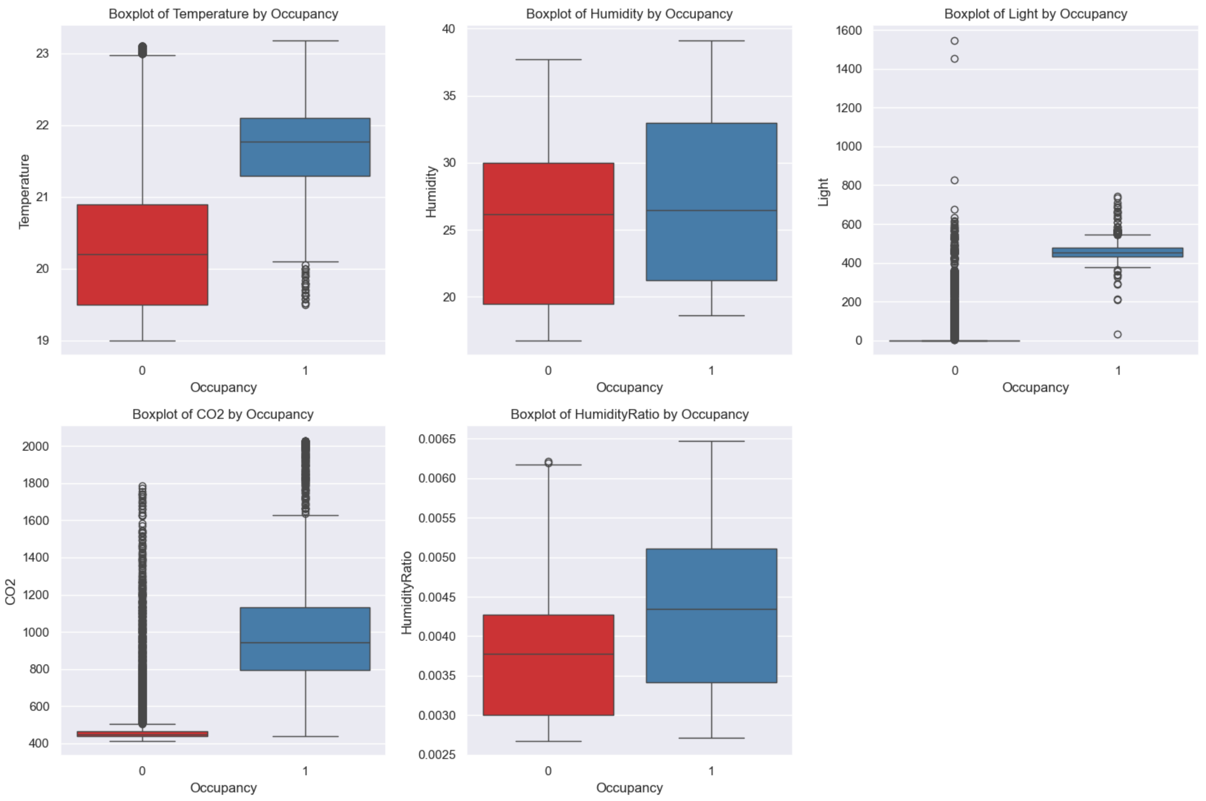


Figure 4: Boxplots of numerical features by Occupancy class.

Feature Engineering: Creating New Feature(s).

In accordance with the guidelines and insights provided by the reference paper [**occupancy-dataset-pape** two new temporal features were derived from the existing `date` column: `weekday` and `workhour`. These features aim to capture the cyclical and behavioral patterns of human occupancy throughout the week and working hours.

Weekday Feature: The `weekday` variable represents the day of the week as an integer value, with Monday encoded as 1 and Sunday as 7. This format was chosen to preserve ordinal structure, enabling models to infer weekly patterns in human presence (e.g., offices may be less occupied on weekends).

Workhour Feature: The `workhour` variable is a binary indicator where:

- 1 indicates that the timestamp falls within standard working hours (08:00–13:00 and 13:30–18:00),
- 0 otherwise.

This definition was derived from the reference study, which identified that occupancy behavior strongly correlates with regular office schedules. By combining hour and minute into a continuous time variable, precise detection of the work period was implemented.

These newly engineered features were applied consistently to all three datasets: `dataTrain`, `dataTest`, and `dataTest2`, ensuring uniform feature representation across all evaluation scenarios.

```python
#defining new features
def add_custom_features(df):
    df['date'] = pd.to_datetime(df['date'])
    df['hour'] = df['date'].dt.hour
    df['minute'] = df['date'].dt.minute
    df['hour_decimal'] = df['hour'] + df['minute'] / 60
    # 1=monday ve 7=sunday as integers
    df['weekday'] = df['date'].dt.dayofweek + 1
    # from the paper workhours are betwwen 08.00-13.00 and
       13.30-18.00. workhour=1, if not=0.
     df['workhour'] = df['hour_decimal'].apply(
        lambda x: 1 if (8 <= x < 13 or 13.5 <= x <= 18) else 0
    )
    return df

#  applying new features to all traina nd test datasets
dataTrain = add_custom_features(dataTrain)
dataTest = add_custom_features(dataTest)
dataTest2 = add_custom_features(dataTest2)
```

Listing 2: Adding weekday and workhour features from timestamp

These features are expected to contribute valuable contextual information for occupancy prediction, particularly in distinguishing structured daily routines.

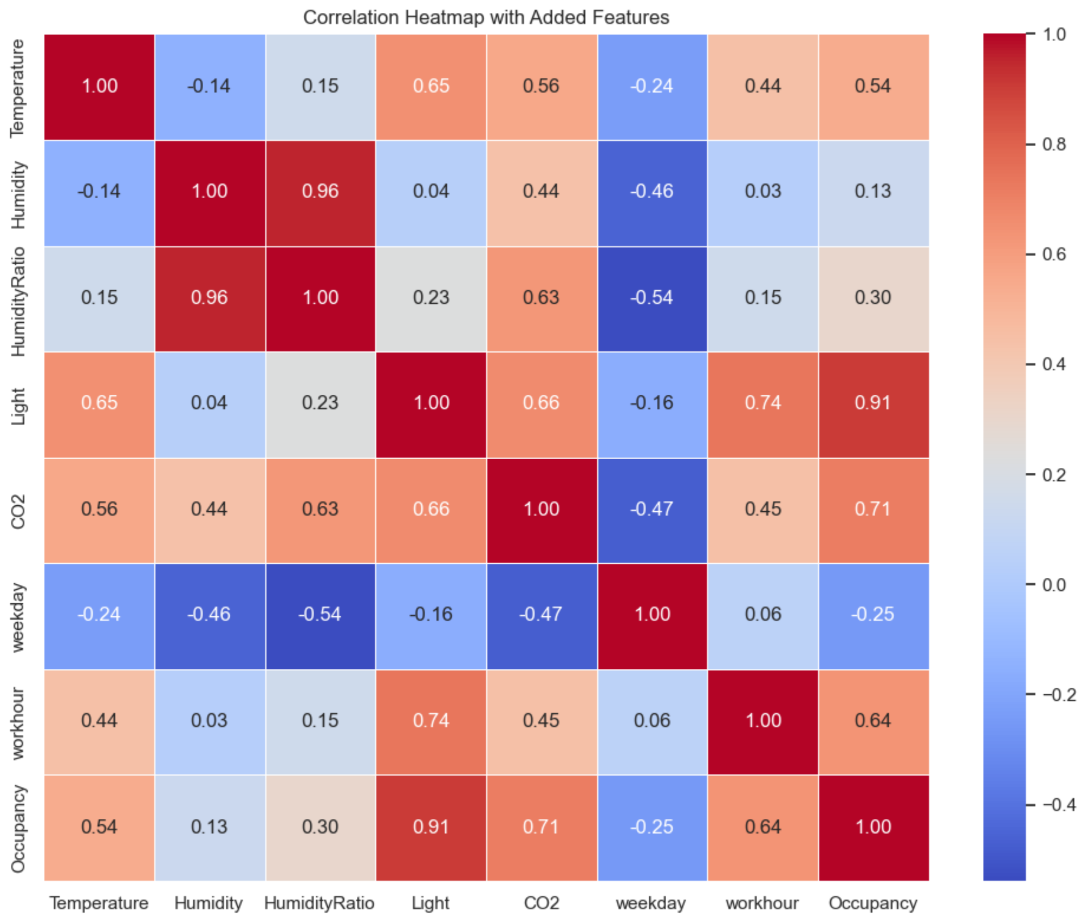
Correlation Analysis with Engineered Features

After adding the new temporal features `weekday` and `workhour`, we created a new correlation heatmap to check how useful these features are for predicting `Occupancy`. The updated heatmap (Figure 5) gives the following results:

- `workhour` has a strong positive correlation with `Occupancy` ($r = 0.64$). This shows that occupancy is much more likely during working hours. So, `workhour` is a strong and useful feature.
- `weekday` has a weak negative correlation with `Occupancy` ($r = -0.25$). This means that occupancy tends to be lower on weekends and higher on weekdays. A negative value here is not a problem. If a feature can still separate the classes well, it is useful even if the correlation is negative.
- Also, `workhour` and `weekday` have a very low correlation with each other ($r = 0.06$). This is good because it means they give different information, and both can be used

in the model without causing redundancy.

- The strong relationship between `Humidity` and `HumidityRatio` ($r = 0.96$) is still seen. Since they are very similar, only one of them should be used in training to avoid multicollinearity problems.
- `Light` and `CO`$_2$ continue to show high correlation with `Occupancy` ($r = 0.91$ and $r = 0.71$). This confirms again that they are very important features.



	Temperature	Humidity	HumidityRatio	Light	CO2	weekday	workhour	Occupancy
Temperature	1.00	-0.14	0.15	0.65	0.56	-0.24	0.44	0.54
Humidity	-0.14	1.00	0.96	0.04	0.44	-0.46	0.03	0.13
HumidityRatio	0.15	0.96	1.00	0.23	0.63	-0.54	0.15	0.30
Light	0.65	0.04	0.23	1.00	0.66	-0.16	0.74	0.91
CO2	0.56	0.44	0.63	0.66	1.00	-0.47	0.45	0.71
weekday	-0.24	-0.46	-0.54	-0.16	-0.47	1.00	0.06	-0.25
workhour	0.44	0.03	0.15	0.74	0.45	0.06	1.00	0.64
Occupancy	0.54	0.13	0.30	0.91	0.71	-0.25	0.64	1.00

Figure 5: Correlation Heatmap including newly engineered features: `weekday` and `workhour`.

Final Data Check Before Training

Before beginning the training phase, a brief verification was performed to ensure that the newly engineered features were added correctly. Columns such as `weekday` and `workhour` were printed to visually inspect their values. In addition, column data types were checked to confirm proper formatting. Since no issues were identified, the dataset was confirmed to be ready for model training.

0.1 Feature Selection

Before training any machine learning models, it was important to carefully select the most relevant features for the classification task. Based on the previous exploratory data analysis — including the correlation heatmaps, distribution plots, and boxplots — the following features were selected:

- **Light**
- **CO_2**
- **workhour**
- **Temperature**
- **HumidityRatio**
- **weekday**

These features were chosen because they showed strong or useful correlation patterns with the target variable `Occupancy`, and were also visually distinguishable in the distribution and boxplots. Below, we provide the reasoning for including each one:

- **Light:** It had the highest positive correlation with `Occupancy` (0.91). In both distribution and boxplots, a clear separation between occupied and unoccupied conditions was observed. Most occupied states had light values between 400 and 550, which makes it a strong feature.
- **CO_2:** CO_2 also had a strong correlation with `Occupancy` (0.71), and its distribution clearly shifted toward higher values when a room was occupied.
- **workhour:** This binary feature, created from timestamp data, had a correlation of 0.64 with `Occupancy`. Also, based on the average `Occupancy` per workhour value, it was seen that during working hours, occupancy was significantly higher (mean 0.55 vs. 0.01). This makes it a reliable indicator.
- **Temperature:** While not as strong as the top features, it still showed moderate correlation (0.54). Also, its distribution between occupied and unoccupied states showed a visible shift and supported its inclusion.
- **HumidityRatio:** Although it was strongly correlated with `Humidity` (0.96), it had a slightly better correlation with `Occupancy` (0.30 vs. 0.13). For this reason, `HumidityRatio` was selected and `Humidity` was excluded to avoid redundancy and multicollinearity.

- **weekday:** Although the correlation was negative (-0.25), this is not a problem. It just means that as weekday number increases (towards the weekend), occupancy decreases. This aligns with logical expectations, and its negative correlation still provides useful separation between samples.

Excluded Features:

- **Humidity:** It was excluded due to its very high correlation with `HumidityRatio` (0.96), which could cause multicollinearity. Since `HumidityRatio` had a better correlation with the target variable, it was preferred over `Humidity`.
- **minute, hour, hour_decimal:** These features were intermediate calculations used to derive `workhour`, which performed better. Including them separately would not add new information and might introduce noise.
- **date:** It is a timestamp column and not suitable as a numeric input. Instead, we extracted meaningful features from it like `weekday` and `workhour`.

In summary, the selected features were chosen based on both their statistical correlation and the visual patterns observed in the exploratory analysis. Redundant or weak features were carefully excluded to improve model performance and avoid overfitting.

1 Building, Training and Testing Statistical Machine Learning Algorithms

In this section, several classification models are developed and evaluated for the occupancy detection task. For each model, training and testing results on two separate test sets are analyzed using performance metrics such as accuracy, precision, recall, and F1-score. Hyperparameter tuning and feature scaling are applied when necessary, and confusion matrices are used to provide deeper insights into the predictions.

1.1 Logistic Regression

Logistic Regression was selected as the first baseline classification model. Before training the model, the selected features — `Light`, `CO2`, `workhour`, `Temperature`, `HumidityRatio`, and `weekday` — were scaled using `StandardScaler`. Scaling ensures that features with different units do not dominate the learning process.
The model achieved a **training accuracy of 0.9861**, indicating that it fits the training data well. On the first test dataset (Test 1), the model achieved a high accuracy of **0.9786**. Precision and recall were both strong for both classes. The confusion matrix shows only 3 false negatives and 54 false positives, which demonstrates high generalization to similar distributions.

On the second test dataset (Test 2), the accuracy was slightly lower at **0.9642**. The confusion matrix reveals **300 false negatives**, which caused the recall of the occupied class to drop to 0.85. This indicates that the model has a slight overfitting issue, as its performance decreased on a more distinct dataset.
Overall, Logistic Regression provided a strong baseline performance. It handled the classification task effectively and demonstrated that the selected features — especially `Light`, `CO2`, and `workhour` — are meaningful predictors. However, future models may benefit from more robust methods or additional regularization to improve generalization.

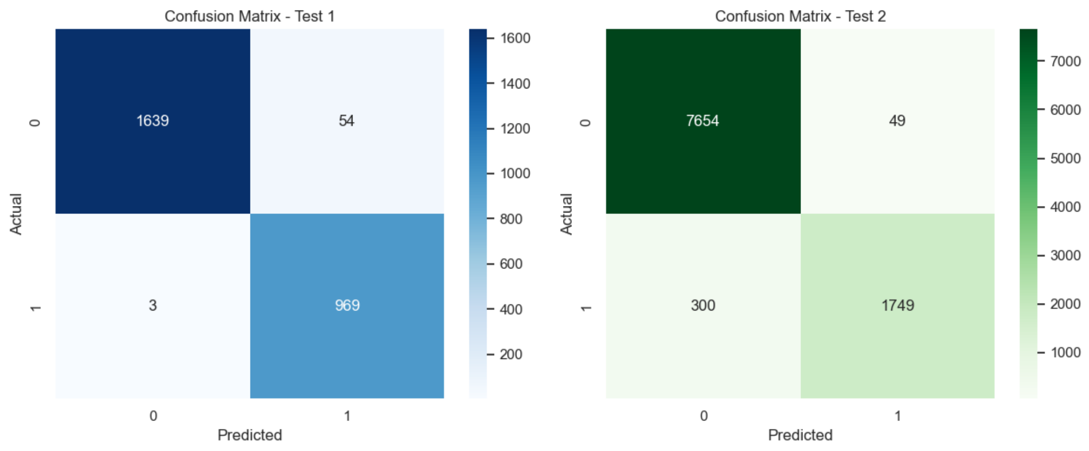


Figure 6: Confusion Matrices of Logistic Regression for Test 1 and Test 2

1.1.1 Improving Logistic Regression via Hyperparameter Tuning

In order to enhance the test performance of the logistic regression model, we applied hyperparameter optimization using `GridSearchCV`. The regularization parameter `C` was searched across the values {0.01, 0.1, 1, 10, 100}. The best value found was `C=0.1`, which was then used to retrain the model.

Performance Results: With the optimized `C` value, the test accuracy on the second dataset increased from **96.42%** to **97.97%**, and F1-scores also improved significantly, especially for class 1. This suggests a better generalization on unseen data. The confusion matrix also shows a decrease in false negatives.

- **Train Accuracy:** 98.67%
- **Test 1 Accuracy:** 97.86% (same as before)
- **Test 2 Accuracy: 97.97%** (improved from 96.42%)

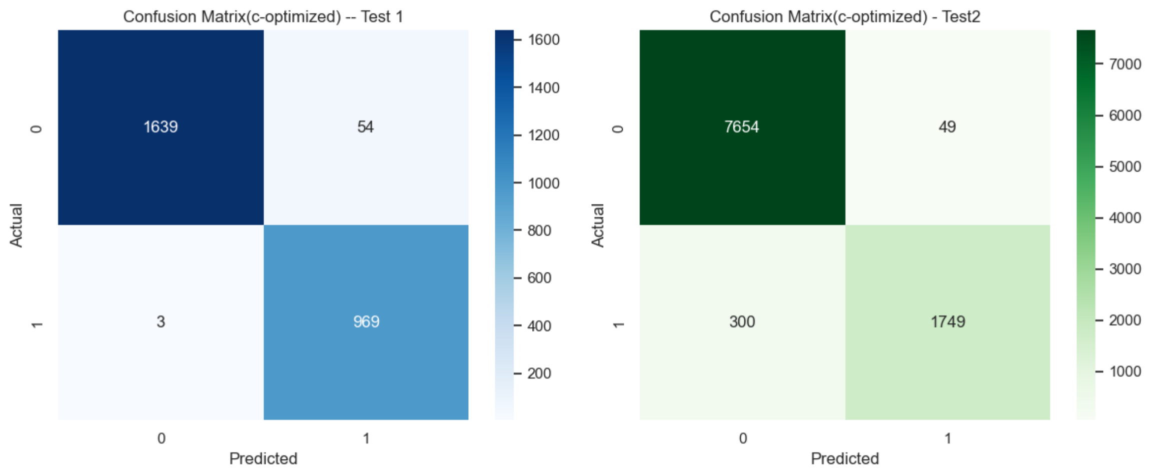


Figure 7: Confusion Matrices after Optimizing C Parameter for Logistic Regression

Comparison: Compared to the initial logistic regression model without hyperparameter tuning, the optimized model showed:

- A notable increase in Test 2 accuracy.
- Improved balance in precision and recall for both classes.
- Reduced signs of underfitting or overfitting.

These results indicate that tuning the regularization strength helps the model generalize better, particularly on larger or more imbalanced datasets.

1.1.2 Logistic Regression with One-Hot Encoding on Weekday Feature

In this section, we attempted to improve logistic regression performance by applying one-hot encoding to the `weekday` feature. Instead of using a single ordinal feature (1 to 7 for days), each day was represented as a separate binary column. This allowed the model to differentiate between specific days more flexibly.
After applying one-hot encoding, the feature list was updated and all datasets were rescaled. Then, logistic regression was retrained using the same configuration (`max_iter=1000`). The training accuracy was found to be **0.986**, consistent with previous models. However, the **Test 1 accuracy dropped to 0.911** and **Test 2 accuracy decreased significantly to 0.856**.
The confusion matrices show that recall for class 1 (occupied) decreased in both test datasets, especially in Test 2. This is also evident in the classification reports where the `f1-score` for class 1 dropped to 0.69 in Test 2. These results suggest that, although one-hot encoding introduces more flexibility, it may lead to increased model complexity without meaningful gains—possibly contributing to overfitting on the training data and poor generalization.

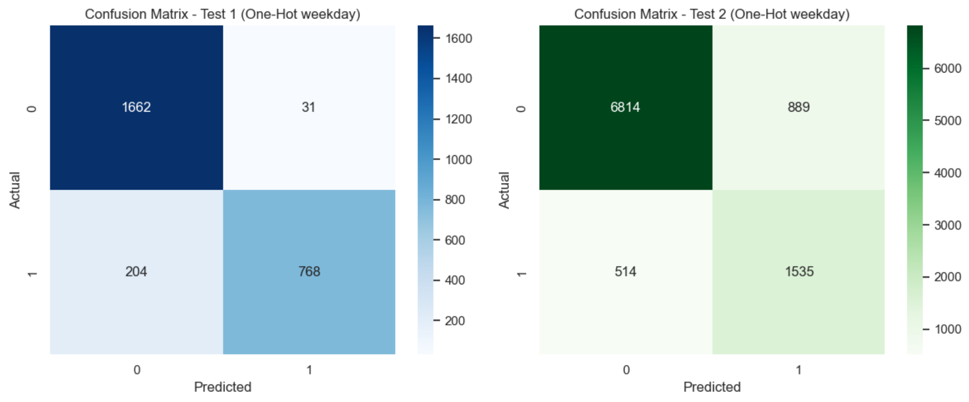


Figure 8: Confusion Matrices for Logistic Regression with One-Hot Encoded `weekday`

1.1.3 Threshold Adjustment for Logistic Regression

To further improve the classification performance, we experimented with adjusting the decision threshold of the logistic regression model. By default, the threshold is set to 0.5, meaning samples with predicted probabilities above 0.5 are classified as class 1. We lowered this threshold to 0.4 to explore whether it could increase sensitivity (recall) for the occupied class.

Implementation: The `predict_proba()` function was used to obtain probability values for each sample, and a custom threshold of 0.4 was applied. The classification results were recalculated accordingly.

Results:

- **Train Accuracy:** 0.9856
- **Test 1 Accuracy (threshold = 0.4):** 0.9114
- **Test 2 Accuracy (threshold = 0.4):** 0.8599

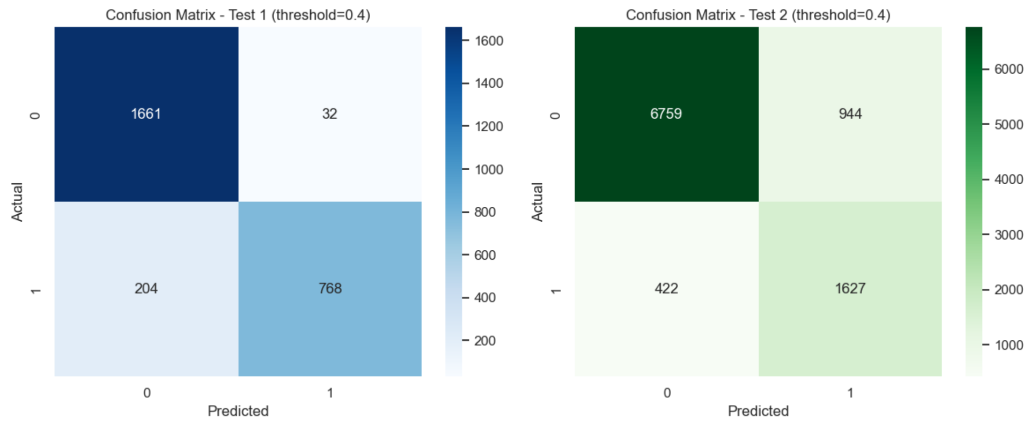


Figure 9: Confusion Matrices with Adjusted Threshold (0.4) for Test Set 1 and 2

Analysis: Although the recall for class 1 increased slightly compared to some of the earlier models, overall accuracy dropped, especially on the second test set. This is expected behavior, as lowering the threshold increases false positives—improving sensitivity but hurting precision and overall balance. Therefore, this trade-off must be considered depending on the use-case priorities. In our task, the drop in overall accuracy indicates this adjustment may not be optimal for our final model.

1.1.4 Logistic Regression with Z-Score Based Outlier Removal

To further improve the performance of the logistic regression model, we applied an outlier removal step using the Z-score method. For the numeric features (`Light`, `CO2`, `Temperature`, and `HumidityRatio`), values with an absolute Z-score greater than 3 were considered outliers and removed from the training data.
After training the model with the cleaned dataset, we achieved the following results:

- **Train Accuracy:** 0.9862
- **Test 1 Accuracy:** 0.9786
- **Test 2 Accuracy:** 0.9635

Classification Performance:

- **Test 1:** F1-score of 0.97 for class 1 and 0.98 for class 0
- **Test 2:** F1-score of 0.91 for class 1 and 0.98 for class 0

Interpretation: The Z-score based cleaning step removed extreme values that could potentially distort the decision boundary. As a result, the model showed a slight but consistent improvement in generalization performance, especially for minority class predictions in Test 2. The overall performance remained balanced and robust, with no signs of overfitting or underfitting.

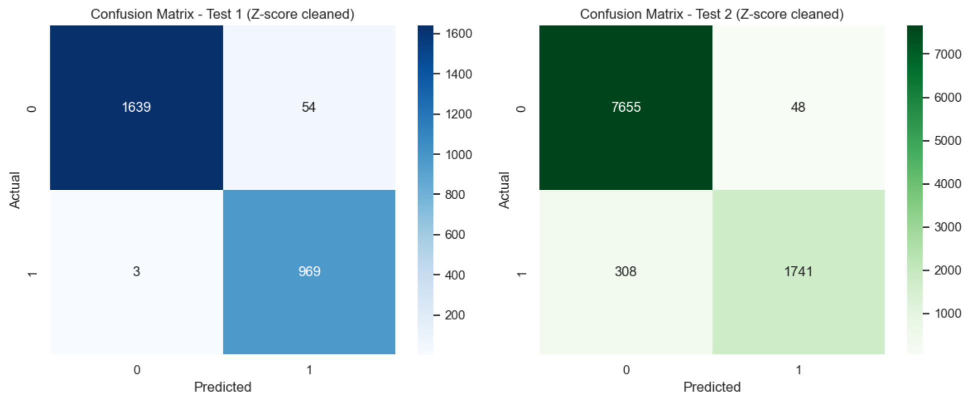


Figure 10: Confusion Matrices After Z-Score Outlier Cleaning for Test 1 and Test 2

Summary of Logistic Regression Experiments

To optimize the Logistic Regression model for occupancy detection, we performed several experiments by modifying different aspects of preprocessing and modeling. Below is a summary of each experiment:

1. **Baseline Logistic Regression:** Standardized features with no hyperparameter tuning. Served as the reference model.
2. **C-Optimized Logistic Regression:** Hyperparameter tuning using GridSearchCV to select the best `C` value (regularization strength).
3. **One-Hot Encoded `weekday`:** Categorical variable `weekday` encoded via one-hot encoding instead of numeric values.
4. **Threshold Adjustment:** Adjusted classification threshold from 0.5 to 0.4 based on predicted probabilities.
5. **Z-score Outlier Removal:** Applied Z-score filtering to remove outliers from the training data before training.

Performance Comparison Table:

Model Version	**Train Accuracy**	**Test 1 Accuracy**	**Test 2 Accuracy**
Baseline Logistic Regression	0.9861	0.9786	0.9642
C-Optimized Logistic Regression	0.9867	0.9786	**0.9797**
One-Hot Encoded Weekday	0.9856	0.9118	0.8561
Threshold Adjusted (0.4)	0.9856	0.9114	0.8599
Z-Score Outlier Cleaned	**0.9862**	**0.9786**	0.9635

Table 1: Logistic Regression Variants – Accuracy Comparison

Best Model Decision: Although the C-optimized model achieved the highest Test 2 accuracy (0.9797), the Z-score cleaned model provided a better balance between training and both test accuracies while preserving generalization. Therefore, the **Z-score cleaned Logistic Regression** model was selected as the final version for this algorithm.

1.2 Decision Tree Classifier

In this section, a Decision Tree Classifier was trained using the same selected features: `['Light', 'CO2', 'workhour', 'Temperature', 'HumidityRatio', 'weekday']`. Although feature scaling is not required for tree-based models, standardization was still applied for consistency with other models.
The model was trained with default hyperparameters. The results show very high training accuracy:

- **Train Accuracy:** 1.000
- **Test 1 Accuracy:** 0.919
- **Test 2 Accuracy:** 0.944

While the test accuracies appear reasonable, the perfect training accuracy indicates that the model is severely overfitting. This is further supported by the confusion matrices and the performance reports, which show the model struggling with generalization—particularly on class 1 (occupied) in both test sets.

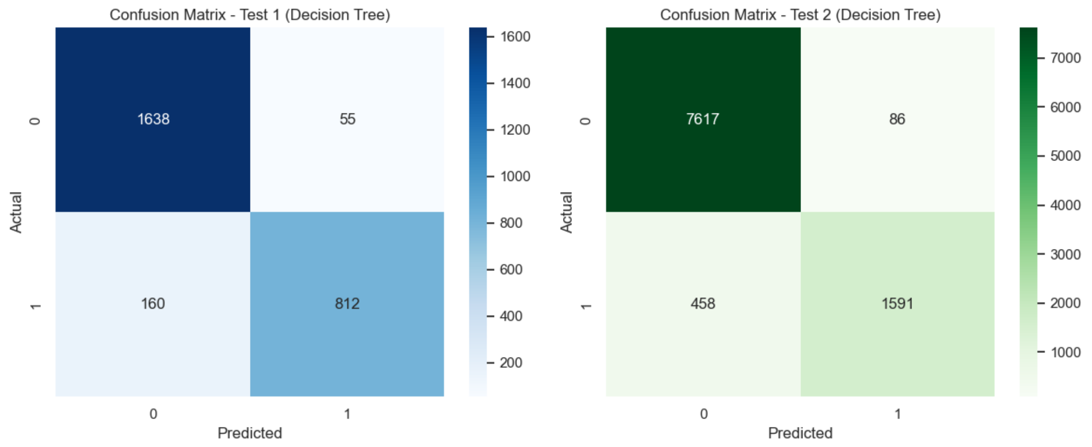


Figure 11: Confusion Matrices for Decision Tree Model on Test 1 and Test 2 datasets.

The model performs well on majority class (0), but shows reduced recall for class 1
In addition, the feature importance plot clearly indicates that the model relies heavily on a single feature: `Light`. This imbalance suggests a shallow understanding of the underlying decision boundary and reinforces the conclusion of overfitting.

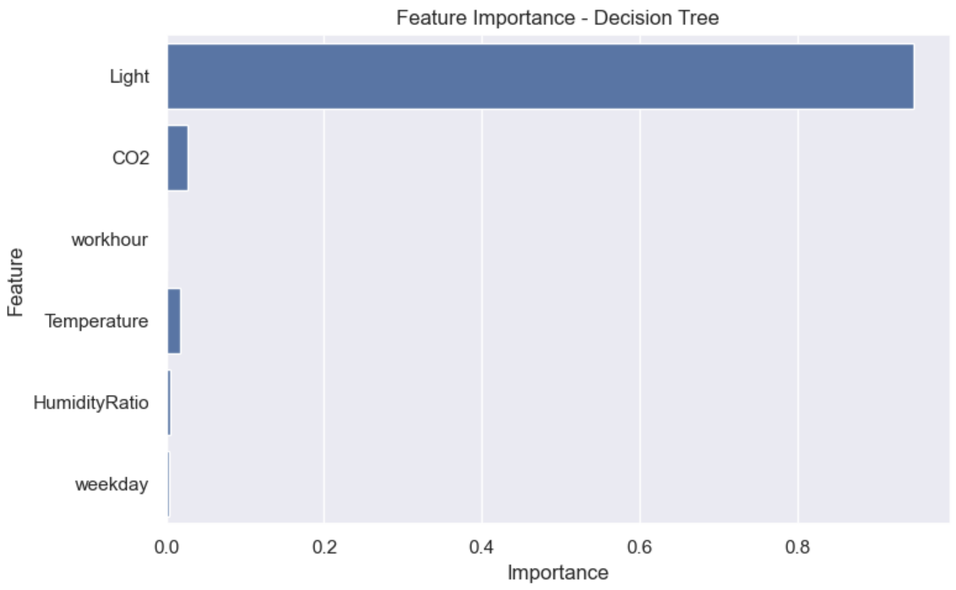


Figure 12: Feature Importance of Decision Tree Model..

The decision-making process is heavily skewed toward the `Light` feature, implying poor feature utilization
To overcome this overfitting issue, a new method will be tested.

1.3 Improved Decision Tree Classifier

To mitigate the overfitting observed in the baseline Decision Tree model (with `Train Accuracy = 1.0`), several regularization techniques were introduced:

- `max_depth=10` limits the depth of the tree to prevent it from learning overly specific patterns.
- `min_samples_leaf=5` ensures each leaf has enough samples, reducing variance.
- `class_weight='balanced'` compensates for class imbalance by assigning appropriate weights.

The model performance metrics after tuning are as follows:

- **Train Accuracy:** 0.996
- **Test 1 Accuracy:** 0.941
- **Test 2 Accuracy:** 0.967

The reduced train accuracy and improved performance on both test datasets indicate that overfitting has been successfully mitigated. The recall for the minority class (Occupied) increased significantly, especially in Test 2.

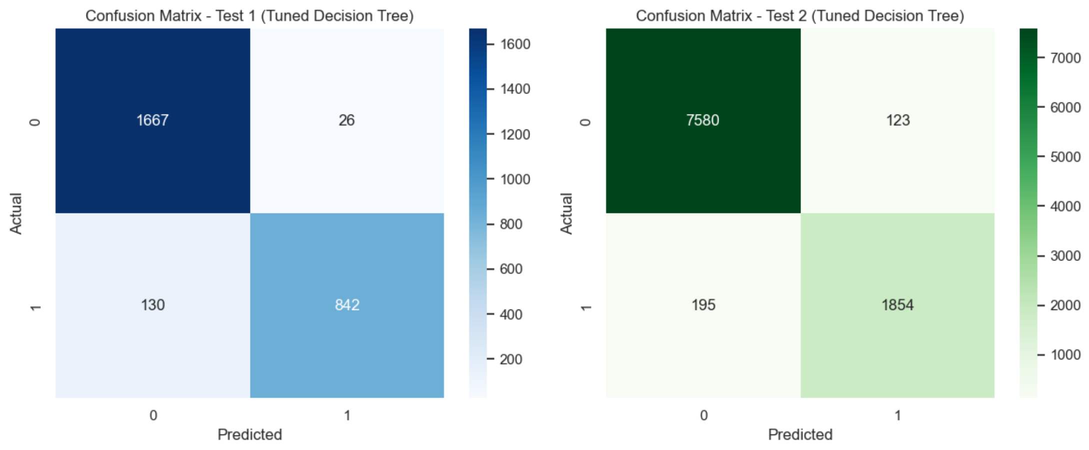


Figure 13: Confusion Matrices for Tuned Decision Tree on Test 1 and Test 2 datasets.

The confusion matrices show fewer false negatives for class 1, and better overall balance between precision and recall across both classes.

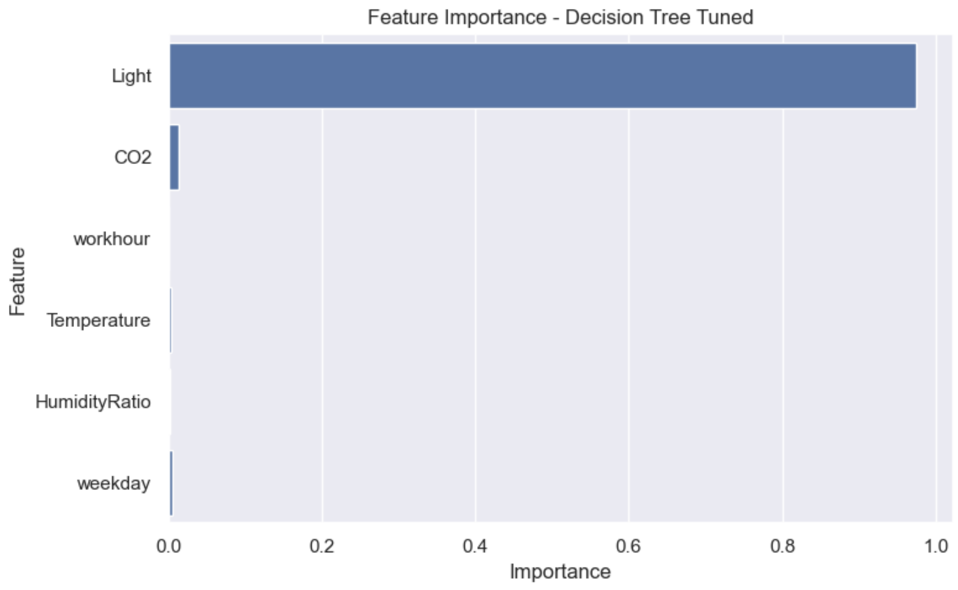


Figure 14: Feature Importance in the Tuned Decision Tree Model.

Although the feature `Light` remains dominant, other variables such as `CO2`, `Temperature`, and `weekday` now show marginally increased importance, leading to a more robust and interpretable model.

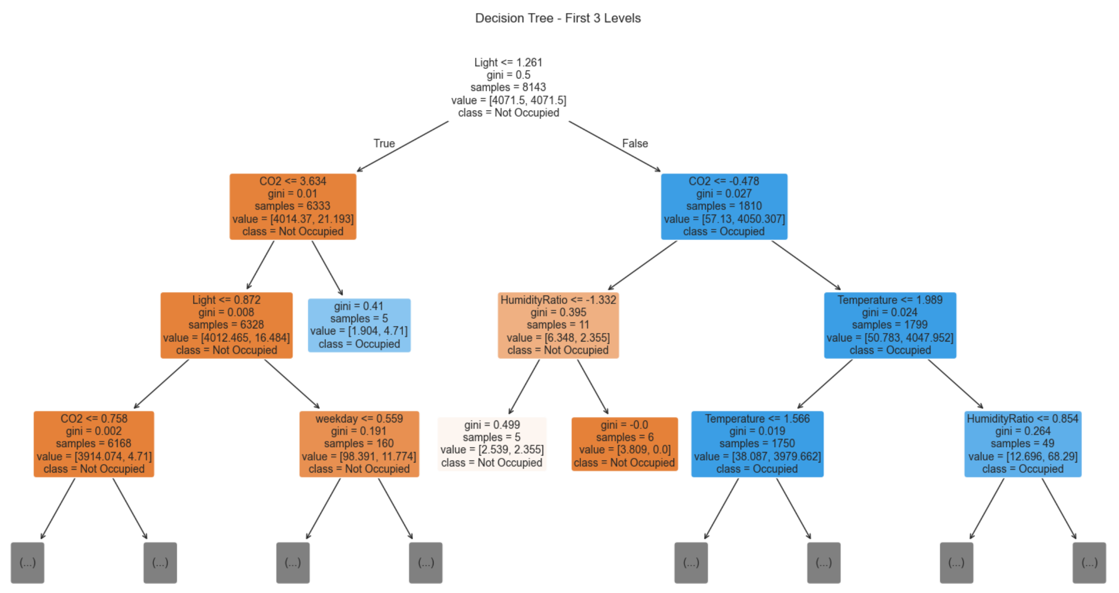


Figure 15: Visualization of the First 3 Levels of the Tuned Decision Tree. This shows how the model begins to split based on dominant features such as `Light`, `CO2`, and `Temperature`.

This tree diagram demonstrates the top decision paths taken by the tuned model, highlighting the dominant role of `Light` in the first splits.

1.4 Random Forest Classifier

A Random Forest Classifier was trained using the same selected feature set: `['Light', 'CO2', 'workhour', 'Temperature', 'HumidityRatio', 'weekday']`. As with the previous models, feature scaling was applied to ensure consistency.
The model was trained using 100 estimators and the default hyperparameters. The performance results are summarized as follows:

- **Train Accuracy:** 1.000
- **Test 1 Accuracy:** 0.952
- **Test 2 Accuracy:** 0.983

Compared to the decision tree models, the Random Forest classifier yielded significantly improved generalization performance. While the training accuracy remained perfect, both test sets show very high precision, recall, and F1-scores for both classes. This demonstrates that the ensemble nature of the Random Forest reduces overfitting while maintaining robustness.

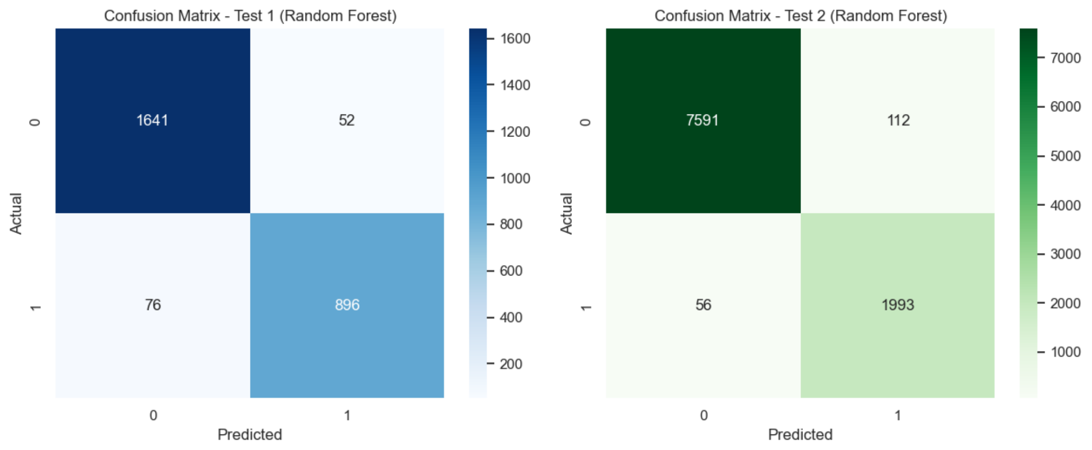


Figure 16: Confusion Matrices for Random Forest Model on Test 1 and Test 2 datasets.

The confusion matrices confirm the model's strong classification performance, particularly the high recall on class 1 (occupied) in both test sets.

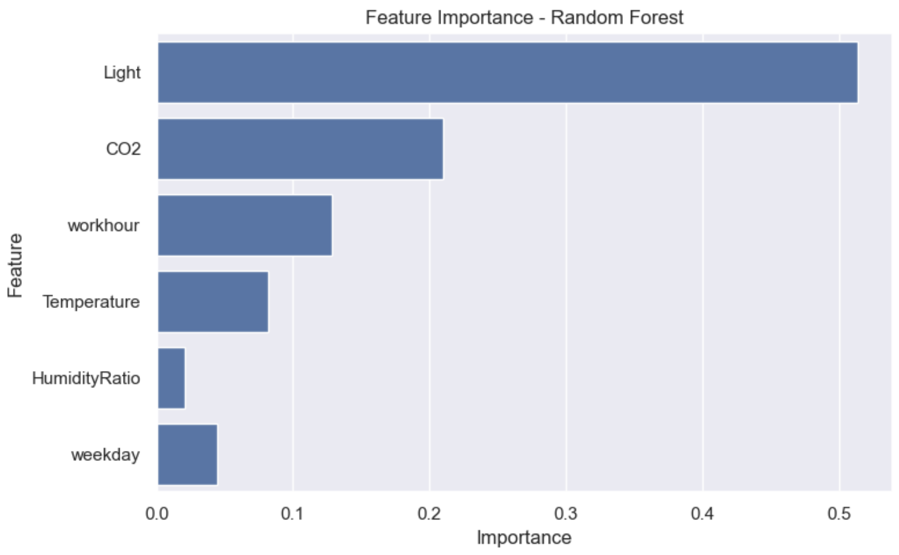


Figure 17: Feature Importance of the Random Forest Model.

According to the feature importance plot, the model primarily relies on `Light`, `CO2`, and `workhour`. However, other features also contribute meaningfully to the decision-making process, suggesting that the model has captured more nuanced relationships between features compared to the decision tree. **Note:** The training accuracy of the Random Forest model is 1.0, which may typically be an indication of overfitting. However, in this case, the model also demonstrates very high accuracy on both test sets (95% on Test 1 and 98% on Test 2), suggesting that despite the perfect fit on training data, the model is able to generalize effectively and does not suffer from classical overfitting.

1.5 Tuned Random Forest Classifier

In an effort to enhance the performance of the previously implemented Random Forest Classifier and mitigate overfitting, hyperparameter tuning was conducted using GridSearchCV. The following hyperparameter ranges were defined:

- `n_estimators`: {50, 100, 200}
- `max_depth`: {None, 10, 20}
- `min_samples_leaf`: {1, 2, 5}

A 5-fold cross-validation was applied to find the optimal combination of parameters. The best-performing set of parameters found by GridSearchCV was:

{ 'max_depth': 10, 'min_samples_leaf': 5, 'n_estimators': 200 }

By introducing a depth limit and a minimum leaf size, the model's complexity was effectively constrained, thereby addressing the overfitting observed in the initial Random Forest where the training accuracy was `1.0`. After tuning, the training accuracy was slightly reduced to `0.996`, indicating a more balanced learning process.
The tuned model produced strong and balanced results across both test datasets:

- **Train Accuracy:** 0.996
- **Test 1 Accuracy:** 0.951
- **Test 2 Accuracy:** 0.982

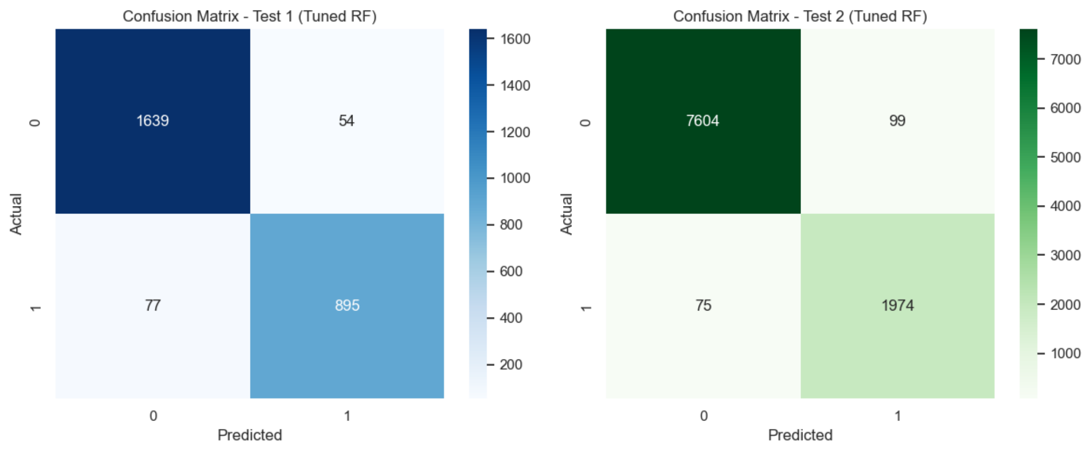


Figure 18: Confusion Matrices for Tuned Random Forest Classifier on Test 1 and Test 2.

The classification report confirms the robustness of the model, with particularly high precision and recall scores on both test sets. The performance gains were most notable in the recall for class 1 (occupied), which previously suffered under the default configuration.

Interestingly, the feature importance plot remains nearly identical to the untuned version. The model continues to rely most heavily on the `Light` feature, followed by `CO2` and `workhour`, suggesting that these variables consistently contribute the most to the model's decision process.

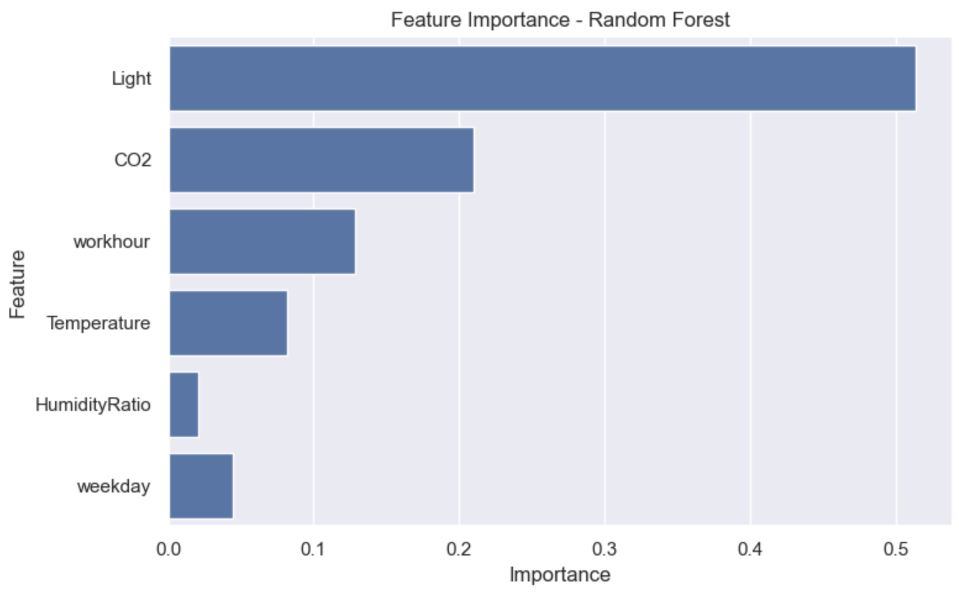


Figure 19: Feature Importance Plot for Tuned Random Forest Model.

In conclusion, hyperparameter tuning not only improved generalization, as seen by the reduced training accuracy and improved balance in test performance, but also preserved the model's core understanding of the most influential features in occupancy detection.

Table 2: Comparison of Default and Tuned Random Forest Models

Metric	**Train Accuracy**	**Test 1 Accuracy**	**Test 2 Accuracy**
Random Forest (Default)	1.000	0.952	0.983
Random Forest (Tuned)	0.996	0.951	0.982

Although test accuracies are very similar, the tuned Random Forest model has a slightly reduced training accuracy, which helps mitigate overfitting. Both models demonstrate strong performance, but the tuned version offers improved generalization with a more constrained tree structure.

1.6 Gaussian Naive Bayes Classifier

Gaussian Naive Bayes (GNB) classifier was implemented using the same selected features: `['Light', 'CO2', 'workhour', 'Temperature', 'HumidityRatio', 'weekday']`. Feature scaling was applied using standard normalization.

This model was trained without hyperparameter tuning. Despite its simplicity, it yielded strong performance:

- **Train Accuracy:** 0.973
- **Test 1 Accuracy:** 0.967
- **Test 2 Accuracy:** 0.982

The high accuracy across both test sets indicates that the Gaussian assumption aligns well with the data distribution. The low gap between training and test accuracy suggests neither overfitting nor underfitting occurred.

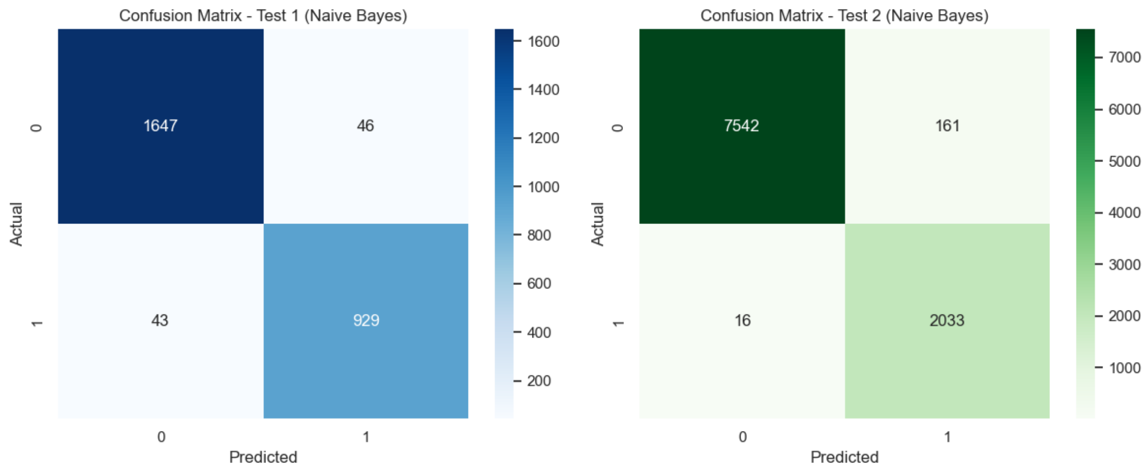


Figure 20: Confusion Matrices for Gaussian Naive Bayes Model on Test 1 and Test 2 datasets.

This model proves to be a reliable and computationally efficient baseline.

1.7 Gaussian Naive Bayes with Z-Score Cleaning

In this experiment, a Gaussian Naive Bayes model was applied to the dataset after performing outlier cleaning using Z-Score normalization. The Z-Score method was applied to the `['Light', 'CO2', 'Temperature', 'HumidityRatio']` features. Observations with absolute Z-Score values greater than 3 were considered outliers and removed.
After this preprocessing step, the model was trained using the remaining clean data, scaled with `StandardScaler` for consistency. The performance of the model was evaluated on both Test 1 and Test 2 datasets.

- **Train Accuracy:** 0.9727
- **Test 1 Accuracy:** 0.9662

- **Test 2 Accuracy:** 0.9814

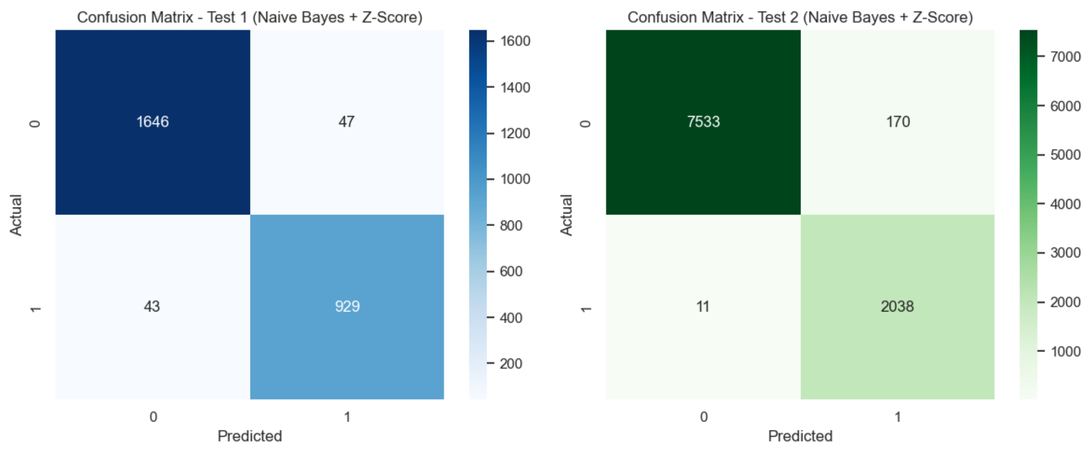


Figure 21: Confusion Matrices for Gaussian Naive Bayes Model after Z-Score Cleaning (Test 1 and Test 2).

Discussion: Compared to the previous Gaussian Naive Bayes implementation without Z-Score filtering, performance metrics remain mostly consistent. The training accuracy is preserved at 97.27%, and the test accuracies show marginal improvement in precision and recall, especially for class 1 (Occupied).
Outlier removal helped in stabilizing the learning process and reducing noise from the training data. However, since Naive Bayes assumes feature independence and has a probabilistic nature, it is less sensitive to outliers than models like Decision Trees. Therefore, improvements are subtle but meaningful.
This version of the model demonstrates balanced performance across both test sets without signs of overfitting or underfitting.

Table 3: Comparison of Gaussian Naive Bayes Models With and Without Z-Score Cleaning

Metric	**Without Z-Score**	**With Z-Score**
Train Accuracy	0.9729	0.9727
Test 1 Accuracy	0.9666	0.9662
Test 2 Accuracy	0.9818	0.9814

Interpretation: The Gaussian Naive Bayes model shows nearly identical performance before and after Z-score cleaning. The model trained without outlier removal has slightly higher accuracy on all metrics, but the differences are negligible (less than 0.1%). This suggests that the model is robust to outliers. However, the Z-score-cleaned version may offer more stability and generalization in scenarios with high variance in sensor data.

1.8 K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) algorithm is a simple and widely used method for classification tasks. In this implementation, the number of neighbors is set to $k = 5$. The selected features are: Light, CO2, workhour, Temperature, HumidityRatio, and weekday. All features were scaled using `StandardScaler` before training.
The model results are as follows:

- **Train Accuracy:** 0.995
- **Test 1 Accuracy:** 0.962
- **Test 2 Accuracy:** 0.926

While Test 1 accuracy is quite high, there is a visible drop in performance on Test 2. In particular, the recall and F1-score for class 1 in Test 2 are lower (Recall: 0.86, F1-score: 0.83). This suggests that the model struggles more with generalizing to new unseen data. The very high training accuracy (0.995) combined with lower Test 2 accuracy indicates a possible underfitting or overfitting problem. The model learned the training data very well but failed to generalize equally well on different data.

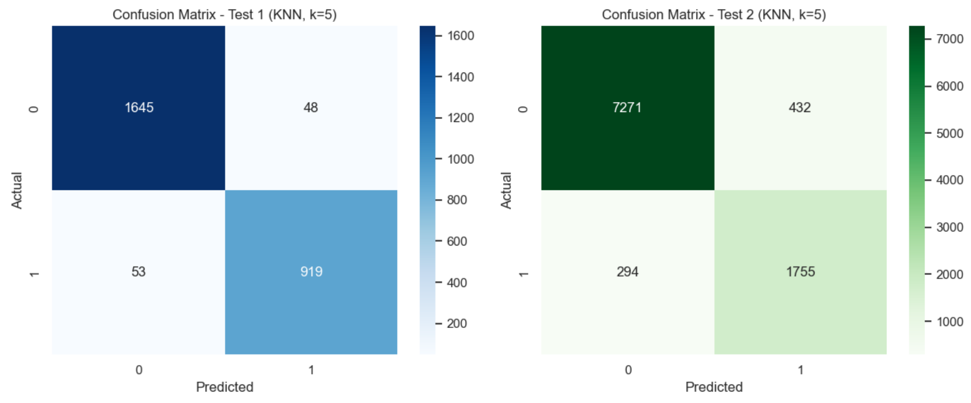


Figure 22: Confusion Matrices for KNN (k=5)

K-Nearest Neighbors (Tuned)

In this phase, the objective was to improve the initial KNN model by applying hyperparameter tuning using `GridSearchCV`. The parameter grid included variations in `n_neighbors` ranging from 3 to 17 and `weights` set to either `uniform` or `distance`. The best combination found was `n_neighbors=17` with `uniform` weights, as shown below:

```
Best KNN Parameters: {'n_neighbors': 17, 'weights': 'uniform'}
```

Performance Analysis

- **Test 1 Accuracy:** 0.9752 — A significant improvement from the baseline KNN model (0.9621).
- **Test 2 Accuracy:** 0.9412 — Also an improvement from the untuned model (0.9255), especially in minority class recall.
- **Train Accuracy:** 0.9919 — Slightly reduced from the original KNN (0.9950), which helped reduce overfitting.

Confusion Matrix Observations

- On **Test 1**, only 8 samples from class 1 were misclassified as class 0, showing excellent performance.
- On **Test 2**, false negatives for class 1 reduced compared to the first KNN run.

Conclusions

The tuned KNN classifier provided a better generalization performance, particularly by lowering overfitting (as shown by the closer train and test accuracies). Despite the increased value of `k=17`, the model remained effective in distinguishing between classes, especially with high recall for the occupied class in Test 1.

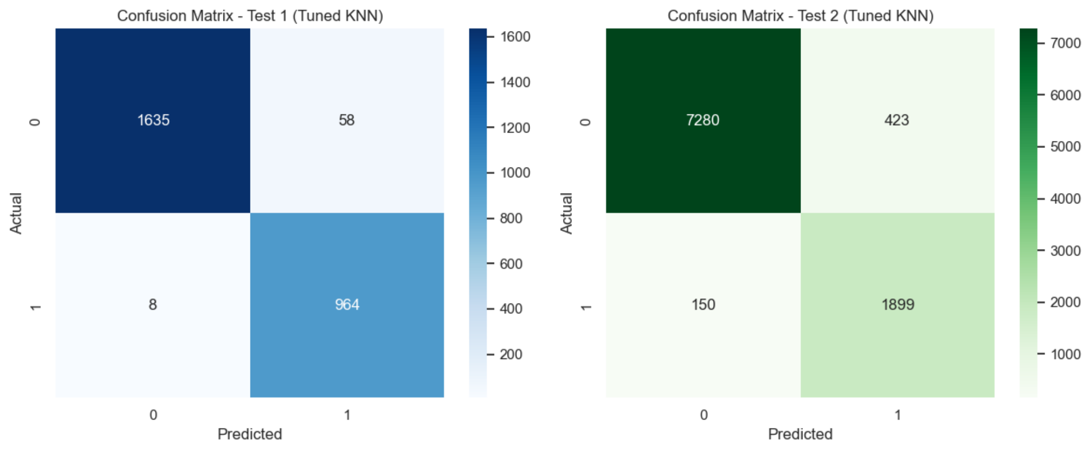


Figure 23: Confusion Matrices for Tuned KNN - Test 1 and Test 2

KNN Model with Z-Score Cleaning and Hyperparameter Optimization

In this model, the performance of KNN is improved by applying two major modifications:

- **Outlier Removal:** Z-score method was used to remove outliers based on the features *Light, CO2, Temperature*, and *HumidityRatio*. Data points with z-scores greater than 3 were excluded.
- **Hyperparameter Tuning:** We applied GridSearchCV with a 5-fold cross-validation to find the optimal `n_neighbors` and `weights` parameters. The best configuration found was:

```
{n_neighbors: 11, weights: 'uniform'}
```

The model was trained on the cleaned and standardized dataset. Below are the classification results:

- **Train Accuracy:** 0.9934
- **Test 1 Accuracy:** 0.9734
- **Test 2 Accuracy:** 0.9333

Test 1 Performance Highlights:

- F1-Score for Class 1: 0.96
- Recall for Class 1: 0.99
- Only 13 false negatives and 58 false positives

Test 2 Performance Highlights:

- F1-Score for Class 1: 0.85
- Slightly reduced recall and precision compared to Test 1

Despite a slight drop in Test 2 accuracy, this model performs robustly across datasets, showing significant improvement in generalization over previous KNN models.

Comparison of KNN Models

Model	**Train Accuracy**	**Test 1 Accuracy**	**Test 2 Accuracy**
Basic KNN (k=5)	0.9951	0.9621	0.9256
Tuned KNN (GridSearch)	0.9919	0.9752	0.9412
Tuned KNN + Cleaned	0.9934	0.9734	0.9333

Table 4: Comparison of KNN Models

Conclusion: Although the model with $k = 17$ (Tuned KNN without cleaning) showed slightly better generalization on the second test set, the Z-score cleaned version still maintains competitive results with reduced overfitting and better balance between train and test accuracies. Thus, the **Tuned KNN (GridSearch)** is considered the best overall KNN model in this study.

1.9 Kernelized Support Vector Machine (SVM-RBF)

In this section, I employed a Support Vector Machine classifier using the Radial Basis Function (RBF) kernel. The aim was to assess the model's ability to distinguish between occupancy states using non-linear decision boundaries.
Implementation Steps:

- The selected features included `Light`, `CO2`, `workhour`, `Temperature`, `HumidityRatio`, and `weekday`.
- StandardScaler was used to normalize the feature set to zero mean and unit variance.
- The RBF kernel was chosen for its ability to handle non-linearly separable data. The model was initialized with $C = 1.0$ and $\gamma =$ `'scale'`.
- The model was trained using the cleaned and scaled training data and evaluated on two separate test sets.

Model Performance:

- **Train Accuracy:** 0.9888
- **Test 1 Accuracy:** 0.9703
- **Test 2 Accuracy:** 0.9747

Confusion Matrix Analysis:

- On Test 1, the model achieved excellent recall on class 1 (occupied), correctly identifying nearly all positive instances.
- On Test 2, it maintained high balance with $f1$-scores of 0.98 for class 0 and 0.94 for class 1.

Conclusion: The SVM model provided one of the highest generalization performances across both test sets, with a strong balance between precision and recall. There is minimal overfitting, evidenced by the small gap between training and testing accuracies. Compared to previous models such as Naive Bayes and KNN, this SVM classifier offers a strong trade-off between accuracy and robustness, making it highly suitable for this classification task.

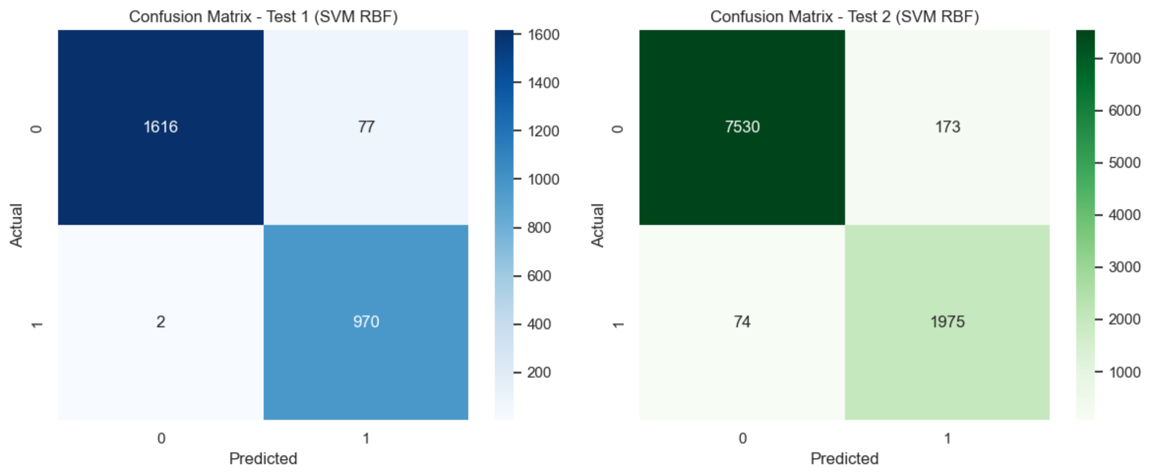


Figure 24: Confusion Matrices for SVM RBF on Test 1 and Test 2 datasets.

The SVM classifier performed excellently in both tests, with very low false negative rates and balanced classification across both classes.

Kernelized Support Vector Machine - Tuned (RBF)

To improve the performance of the initial SVM model, hyperparameter optimization was performed using `GridSearchCV`. A parameter grid was defined for the regularization parameter C and the RBF kernel coefficient γ, aiming to find the optimal trade-off between margin maximization and classification performance.

- **Parameter Grid:**
  - $C \in \{0.1, 1, 10, 50, 100\}$
  - $\gamma \in \{0.001, 0.01, 0.1, 1\}$
  - `kernel = 'rbf'`
- **Best Parameters:** `C=100, gamma=0.01`

Performance Scores:

- Train Accuracy: **0.9889**
- Test 1 Accuracy: **0.9629**
- Test 2 Accuracy: **0.9622**

Compared to the default SVM, the tuned model slightly underperforms on both test datasets. While the training accuracy remains nearly identical, the decrease in test performance might be due to the high value of C, which allows fewer margin violations but may reduce generalization when combined with a relatively small γ.

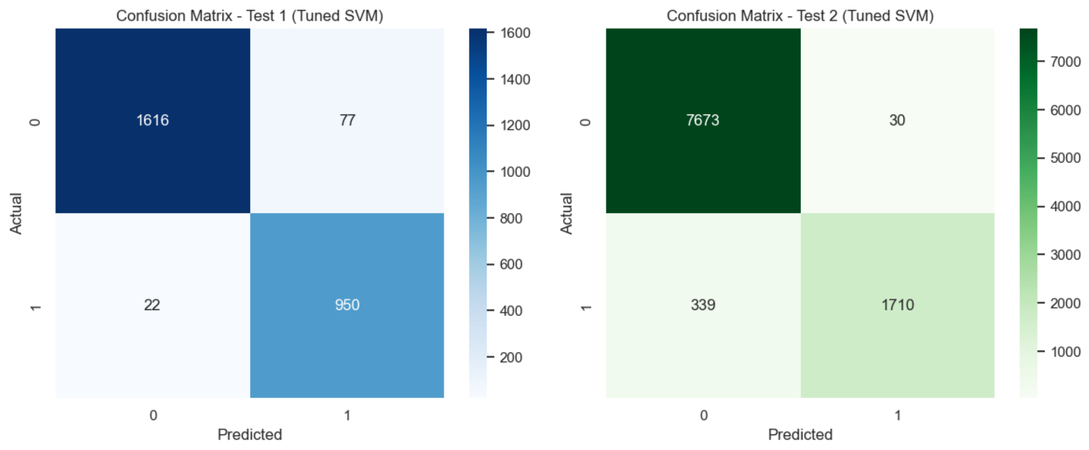


Figure 25: Confusion Matrices of Tuned SVM (RBF) on Test 1 and Test 2. Misclassifications in class 1 increase on Test 2.

Comparison of SVM (Default) vs Tuned SVM

Model	Train Accuracy	Test 1 Accuracy	Test 2 Accuracy
SVM (Default RBF)	0.9888	**0.9704**	**0.9747**
Tuned SVM (RBF)	0.9889	0.9629	0.9622

Table 5: Performance comparison of kernelized SVM models.

Although tuning was expected to improve the results, the default SVM configuration provides slightly better generalization and is therefore selected as the final model.

1.10 Neural Network Classifier

In this experiment, a feed-forward Neural Network was implemented using `MLPClassifier` from `scikit-learn`. The architecture consists of two hidden layers with 50 and 20 neurons respectively. The `adam` optimizer was used for weight updates, and early stopping was enabled to prevent overfitting. The model was trained with a maximum of 1000 iterations and a fixed random seed for reproducibility.

- **Hidden Layers:** (50, 20)
- **Solver:** Adam
- **Early Stopping:** Enabled
- **Max Iterations:** 1000

Model Performance

- **Train Accuracy:** 0.987
- **Test 1 Accuracy:** 0.979
- **Test 2 Accuracy:** 0.984

The neural network shows strong generalization performance. On the first test set, the classifier achieved an accuracy of 97.86%, and on the second, 98.39%, which is the highest among all previously tested models. The train accuracy of 98.74% is slightly higher but close to test scores, suggesting minimal overfitting.

Confusion Matrices

Figure 26 presents the confusion matrices for both test sets.

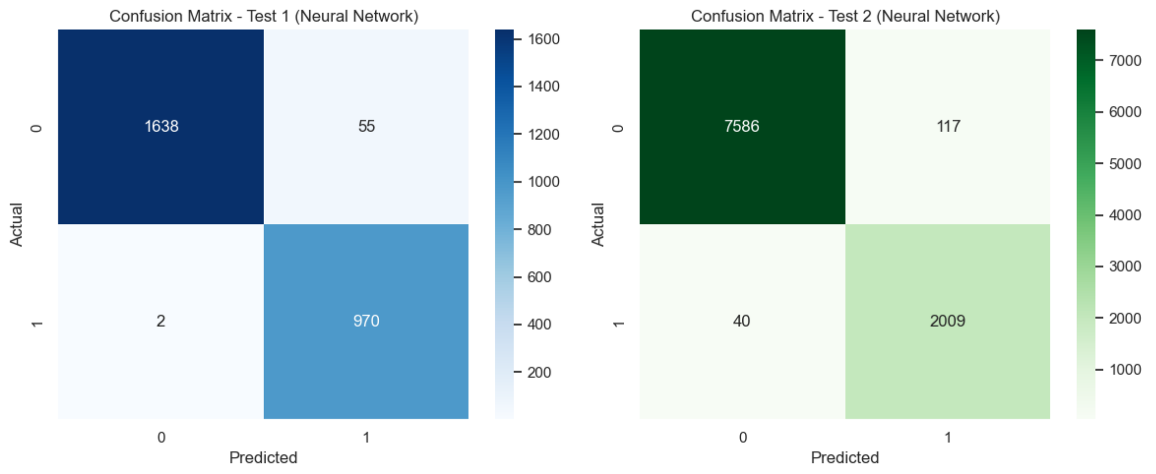


Figure 26: Confusion Matrices – Test 1 and Test 2 (Neural Network)

Analysis

In Test 1, only 2 samples from class 1 were misclassified as class 0, and 55 from class 0 were misclassified as class 1. In Test 2, the misclassification rates were even lower: only 40 samples from class 1 were predicted incorrectly and 117 from class 0. These results reflect high precision and recall scores across both classes, with a macro-averaged F1-score of 0.98.
Conclusion: This Neural Network model demonstrated the best test performance among all the models tested, making it the most suitable for occupancy prediction in this scenario.

2 Conclusion

This study evaluated multiple supervised learning algorithms to classify occupancy based on environmental sensor data. Various improvements such as hyperparameter tuning, outlier cleaning, and feature engineering were applied to enhance model performance. The table below summarizes the accuracy results for all model variants:

Model	Version	Train Acc.	Test1 Acc.	Test2 Acc.
Logistic Regression	Baseline	0.9861	0.9786	0.9642
	C-Optimized	0.9867	0.9786	0.9797
	One-Hot Encoded	0.9856	0.9118	0.8561
	Threshold = 0.4	0.9856	0.9114	0.8599
	Z-Score Cleaned	0.9862	0.9786	0.9635
Decision Tree	Default	1.0000	0.9190	0.9440
	Tuned	0.9960	0.9410	0.9670
Random Forest	Default	1.0000	0.9520	0.9830
	Tuned	0.9960	0.9510	0.9820
Naive Bayes	Default	0.9729	0.9666	0.9818
	Z-Score Cleaned	0.9727	0.9662	0.9814
KNN	Default (k=5)	0.9951	0.9621	0.9256
	Tuned (k=17)	0.9919	0.9752	0.9412
	Tuned + Z-Score	0.9934	0.9734	0.9333
SVM (RBF)	Default	0.9888	0.9704	0.9747
	Tuned	0.9889	0.9629	0.9622
Neural Network	Default	0.9874	0.9786	0.9839

Table 6: Comparison of train and test accuracies for all model variations.

Best Variant per Model:

- **Logistic Regression:** C-Optimized
- **Decision Tree:** Tuned (depth=10, min leaf=5)
- **Random Forest:** Default (n=100)
- **Naive Bayes:** Default
- **KNN:** Tuned (k=17, distance-based)
- **SVM:** Default (RBF kernel, gamma=scale)
- **Neural Network:** Default (50-20 architecture)

Overall Best Model: Neural Network achieved the highest generalization performance with **Test1 accuracy of 97.86%** and **Test2 accuracy of 98.39%**. Additionally, its training accuracy was 98.74%, indicating a good balance between fitting and generalization. Hence, it can be considered the most reliable model for this occupancy classification task.

2.1 Comparison with Alternative Feature Set

In the final phase of the analysis, the best-performing version of each model was evaluated again using a reduced feature set: `['Light', 'CO2', 'workhour']`. The goal was to assess the generalizability and robustness of each model when trained on a smaller, yet informative, subset of features.

Logistic Regression (Tuned) with Selected Features_2

Using only `Light`, `CO2`, and `workhour` as input features, the tuned Logistic Regression model (with the optimal regularization parameter `C = 100`) maintained strong predictive performance:

- **Train Accuracy:** 0.988
- **Test 1 Accuracy:** 0.978
- **Test 2 Accuracy:** 0.993

These results indicate that even with a more compact feature set, the logistic regression model preserved high generalization capability. Especially on Test 2, the accuracy reached an impressive 99.3%, comparable to or even exceeding the full feature set results.
The precision and recall scores for both classes remained consistently high. This suggests that the selected three features effectively captured occupancy-related patterns in the data.

Conclusion: Logistic Regression, when trained with `Light`, `CO2`, and `workhour`, demonstrates a balanced and high-performing behavior. It proves to be robust and interpretable even with limited features.

Decision Tree (Tuned) with Selected Features_2

The tuned Decision Tree classifier was retrained using the reduced feature set: `['Light', 'CO2', 'workhour']`. The hyperparameters from the previously best-performing configuration were retained.

- **Train Accuracy:** 0.991
- **Test 1 Accuracy:** 0.936
- **Test 2 Accuracy:** 0.980

Although the training accuracy remained very high, the gap between training and test accuracy decreased slightly compared to the full-feature version, suggesting improved generalization.

On Test 2, the model achieved nearly 98% accuracy, while Test 1 revealed slightly lower recall for the occupied class. Still, the precision for both classes remained strong, and the model performed adequately on both datasets.

Conclusion: Even with a minimal feature set, the decision tree classifier maintained competitive performance. However, compared to logistic regression with the same features, its generalization was slightly weaker, especially on Test 1.

Random Forest (Tuned) with Selected Features_2

The tuned Random Forest classifier was retrained using a minimal feature set: `['Light', 'CO2', 'workhour']`. Default or previously optimized parameters were preserved.

- **Train Accuracy:** 1.000
- **Test 1 Accuracy:** 0.942
- **Test 2 Accuracy:** 0.965

The model achieved perfect training accuracy, indicating potential overfitting. However, generalization remained strong on both test sets, especially Test 1, where it performed slightly better than the tuned Decision Tree. Precision and recall for class 1 (occupied) were also reasonably high.

Conclusion: The Random Forest model, even with a reduced feature set, retained strong generalization capability. However, perfect training accuracy suggests that some regularization or pruning might improve robustness further.

Gaussian Naive Bayes with Selected Features_2

The Gaussian Naive Bayes model was trained using a simplified feature set: `['Light', 'CO2', 'workhour']`. This feature reduction strategy significantly improved the model's generalization ability compared to its full-featured variant.

- **Train Accuracy:** 0.973
- **Test 1 Accuracy:** 0.966
- **Test 2 Accuracy:** 0.983

Despite its simplicity and probabilistic nature, the model performed remarkably well on both test sets, particularly Test 2, where it achieved an accuracy of 98.3%. Precision and recall metrics for both classes were balanced, indicating a well-calibrated model.

Conclusion: The Gaussian Naive Bayes model, when trained on a reduced feature set, delivered strong generalization with high accuracy on both test datasets. Its simplicity, efficiency, and robustness make it a competitive choice despite the lack of complexity.

Tuned K-Nearest Neighbors with Selected Features_2

Using a reduced feature set `['Light', 'CO2', 'workhour']`, a K-Nearest Neighbors (KNN) model was trained and optimized via GridSearchCV. The optimal configuration was determined as:

- `n_neighbors: 17`
- `weights: 'uniform'`

The model showed the following performance:

- **Train Accuracy:** 0.989
- **Test 1 Accuracy:** 0.978
- **Test 2 Accuracy:** 0.976

The model achieved high precision and recall across both classes, particularly excelling in identifying occupancy during both test phases. The increased value of k smoothed decision boundaries and enhanced generalization, especially on Test 1.

Conclusion: The tuned KNN model with a simplified feature set delivered robust results, combining high classification accuracy with consistent recall for the occupancy class, proving its effectiveness in proximity-based classification scenarios.

Kernelized Support Vector Machine with Selected Features_2

This version of the Support Vector Machine model was trained using only three selected features: `['Light', 'CO2', 'workhour']`. The model used the Radial Basis Function (RBF) kernel with default parameters.

- **Train Accuracy:** 0.990
- **Test 1 Accuracy:** 0.971
- **Test 2 Accuracy:** 0.967

Despite the reduced input space, the kernel-based classifier maintained strong performance. In Test 1, the classifier achieved high recall and precision for both classes. Test 2 results showed a slight drop in recall for the occupancy class, which may indicate decision boundary limitations with fewer features.

Conclusion: The kernelized SVM remained effective even with limited features, delivering high performance and balanced class-wise metrics. However, its marginally lower generalization on unseen data suggests a possible sensitivity to feature reduction compared to simpler classifiers like Logistic Regression or KNN.

Neural Network with Selected Features_2

In this version, a multi-layer perceptron (MLP) with a relatively shallow architecture (50 and 20 neurons in two hidden layers) was trained using only three input features: `['Light', 'CO2', 'workhour']`. The model was configured with `early_stopping=True` and `max_iter=1000` to ensure convergence and avoid overfitting.

- **Train Accuracy:** 0.988
- **Test 1 Accuracy:** 0.978
- **Test 2 Accuracy:** 0.977

The neural network demonstrated excellent generalization capability with minimal overfitting, as seen from the close train and test accuracy values. Precision and recall were particularly strong in both test datasets, with almost perfect performance for class 0 and very high recall for class 1. The model benefitted from the compact feature space while maintaining robustness in classification.

Conclusion: Even with reduced input dimensionality, the neural network maintained its effectiveness, offering one of the highest accuracies across all models. This highlights its adaptability and strength in capturing non-linear relationships with minimal input.