PREDICTING ROOM OACCUPACY

Problem Statement

We shall be using data from the occupnacy estimation experiment to predict the number of room occupants using the Support Vector Machine Model. The data was stored in the UCI machine learning repository.

The UCI Machine Learning Repository has implemented several data management measures to ensure the protection of privacy for individuals accessing the repository and to safeguard the intellectual property rights of data owners and contributors.

Privacy Protection Measures:

1. **Anonymization and De-identification:** The UCI Repository ensures that datasets are anonymized and de-identified to remove personally identifiable information (PII) from the data. This helps prevent the identification of individuals in the dataset.

2. **Data Aggregation:** Data may be aggregated or aggregated features might be provided instead of raw individual data. This reduces the risk of identifying specific individuals while still allowing researchers to derive insights from the data.

3. **Restricted Access:** The repository might limit access to certain sensitive datasets or require users to register and agree to terms of use before accessing certain datasets. This enables tracking and controlling who accesses the data and for what purpose.

4. **Ethics Guidelines:** The repository might provide guidelines for researchers on ethical data usage, encouraging them to handle data responsibly and avoid any unethical or harmful analyses.

Intellectual Property Rights Protection Measures:

1. **Licensing and Terms of Use:** The UCI Repository often requires data contributors to provide information about the licensing terms for their datasets. This helps ensure that users are aware of how they can use the data and what limitations might apply.

2. **Attribution:** Users are usually required to provide proper attribution to the data owners or contributors when using the datasets. This ensures that the intellectual property rights of the original data creators are acknowledged.

3. **Data Ownership:** The repository might clarify the ownership of the data, specifying whether contributors retain ownership or if they transfer certain rights to the repository. This can help prevent unauthorized use or distribution.

4. **Data Use Agreements:** In some cases, the repository might require users to sign data use agreements that outline how the data can be used and what restrictions apply. This adds an

additional layer of legal protection for both data owners and users.

5. **Sensitive Data Handling:** For datasets with proprietary or sensitive information, the repository might establish stricter access controls or require users to request access directly from the data owners. This ensures that only trusted parties can access the data.

6. **Reporting Violations:** The repository might encourage users to report any violations of licensing terms or intellectual property rights. This helps maintain the integrity of the data and ensures that appropriate action can be taken against violators.

Overall, these data management measures help strike a balance between facilitating research and protecting the privacy of individuals and the intellectual property rights of data owners and contributors in the UCI Machine Learning Repository.

The occupancy estimation experiment took place in a room measuring 6m x 4.6m. The setup involved a star configuration with 7 sensor nodes and one edge node. The sensor nodes communicated wirelessly with the edge node, transmitting data every 30 seconds. Throughout the data collection process, no HVAC systems were in operation.

Five types of non-intrusive sensors were utilized in the experiment: temperature, light, sound, CO2, and digital passive infrared (PIR). Manual calibration was required for the CO2, sound, and PIR sensors. To calibrate the CO2 sensor, a zero-point calibration was performed initially by placing it in a clean environment for over 20 minutes and then activating the calibration pin (HD pin) for more than 7 seconds. The sound sensor, essentially a microphone with an attached variable-gain analog amplifier, produced analog output read in volts by the microcontroller's ADC. The amplifier's gain was adjusted using a potentiometer to achieve maximum sensitivity. The PIR sensor featured two trimpots: one for sensitivity and the other for adjusting the duration of high output after motion detection. Both trimpots were set to their highest values. Sensor nodes S1-S4 were equipped with temperature, light, and sound sensors; S5 contained a CO2 sensor, while S6 and S7 each carried a PIR sensor positioned on the ceiling ledges at an angle to optimize motion detection coverage.

Data collection transpired over a controlled 4-day period with room occupancy ranging from 0 to 3 individuals. Manual recording provided the ground truth for the room's occupancy count.

[^1] Singh,Adarsh Pal and Chaudhari,Sachin. (2023). Room Occupancy Estimation. UCI Machine Learning Repository. https://doi.org/10.24432/C5P605.

SUPPORT VECTOR MACHINE MODEL (SVM) MODEL

Support Vector Machines (SVM) are a unique and distinctive class of machine learning algorithms that offer several key differences from most other algorithms. In the context of the occupancy estimation experiment, SVM can be employed for classification tasks to predict occupancy levels based on sensor data. Here's an explanation of what makes SVM special and how it could be utilized, along with the steps, mathematical representation, and assumptions:

What makes SVM special and different:

1. **Margin Maximization:** SVM is focused on finding the hyperplane that best separates different classes while maximizing the margin between them. The margin is the distance between the hyperplane and the nearest data points of each class. This emphasis on maximizing the margin helps SVM to achieve better generalization and resilience against overfitting.

2. **Kernel Trick:** SVM can efficiently handle non-linearly separable data by transforming the original feature space into a higher-dimensional space using a kernel function. This allows SVM to find a hyperplane that separates the data even when they are not linearly separable in the original space.

3. **Support Vectors:** The data points that are closest to the hyperplane and have the most influence on determining its position are called support vectors. These support vectors define the decision boundary and play a crucial role in classification.

4. **Regularization Parameter (C):** SVM uses a regularization parameter 'C' that balances the trade-off between achieving a wider margin and correctly classifying training examples. Smaller 'C' values emphasize a wider margin, potentially allowing for some misclassifications, while larger 'C' values focus more on correct classification.

Using SVM in the occupancy estimation analysis:

Steps:

1. **Data Preprocessing:** Prepare the sensor data by normalizing and standardizing it to ensure features are on a comparable scale.

2. **Feature Extraction:** Extract relevant features from the sensor data, such as temperature, light, sound, CO2 levels, and PIR sensor readings.

3. **Data Split:** Divide the dataset into training and testing subsets. The training set is used to train the SVM model, while the testing set evaluates its performance.

4. **Kernel Selection:** Choose an appropriate kernel function (e.g., linear, polynomial, radial basis function) based on the nature of the data and potential non-linearity.

5. **Model Training:** Train the SVM model using the training data. The algorithm finds the optimal hyperplane that separates different occupancy levels while maximizing the margin and considering the regularization parameter 'C'.

6. **Model Evaluation:** Use the testing data to evaluate the model's performance. Metrics like accuracy, precision, recall, and F1-score can be used to assess how well the SVM model predicts occupancy.

7. **Decision Function:**

   The decision function of an SVM computes the signed distance of a data point from the decision hyperplane. It determines on which side of the hyperplane a data point lies and

assigns a class label accordingly.

Mathematically: `f(x) = w^T * x + b`

where:

- `f(x)` is the output of the decision function for input data `x`.
- `w` is the weight vector.
- `x` is the input feature vector.
- `b` is the bias term.

If `f(x) > 0`, the data point is classified as one class, and if `f(x) < 0`, it's classified as the other class.

8. **Hyperplane Equation:**

   In a two-dimensional feature space (binary classification), the equation of the decision boundary (hyperplane) is given by:

   Mathematically: `w^T * x + b = 0`

   where:

   - `w` is the normal vector to the hyperplane.
   - `x` is the input feature vector.
   - `b` is the bias term.

9. **Margin Calculation:**

   The margin is the distance between the hyperplane and the nearest data point (support vector) of either class. SVM aims to maximize this margin.

   Mathematically: `margin = 2 / ||w||`

   where:

   - `w` is the weight vector.

10. **Optimization Objective (Hinge Loss):**

    SVM aims to minimize the hinge loss, which is a function that penalizes misclassified points and encourages maximizing the margin.

    Mathematically: `Loss(w) = C * Σ(max(0, 1 - y_i * (w^T * x_i + b))) + 0.5 * ||w||^2`

    where:

    - `C` is the regularization parameter.
    - `y_i` is the true class label of the data point `x_i`.
    - `w` is the weight vector.
    - `x_i` is the input feature vector.
    - `b` is the bias term.

These mathematical functions describe the core concepts of the SVM algorithm, including how it defines the decision boundary (hyperplane), how it calculates the margin, and how it optimizes the model parameters to find the best separation of data points.

Assumptions:

1. **Linear Separability (for linear kernel):** The SVM algorithm assumes that the data can be separated by a linear boundary. If data points are not linearly separable, a kernel function is used to transform the data into a higher-dimensional space where linear separation is possible.
2. **Feature Independence:** SVM assumes that features are independent and contribute equally to the decision boundary. If features are highly correlated, the performance might degrade.
3. **Low Noise:** SVM performs well when the data has low noise levels. Outliers or noisy data points can have a significant impact on the position of the decision boundary.
4. **Binary Classification:** SVM is initially designed for binary classification tasks. For multi-class problems, techniques like one-vs-one or one-vs-rest are used.

In the context of the occupancy estimation experiment, SVM can help classify different occupancy levels based on the sensor data, taking advantage of its unique characteristics of margin maximization and kernel-based non-linear separation.

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_predict, cross_val_sco
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, r
from sklearn.preprocessing import PowerTransformer, OneHotEncoder
from sklearn.compose import ColumnTransformer
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import plotly.figure_factory as ff
```

ANALYSIS AND FINDINGS

In [ ]:
```
df = pd.read_csv("Occupancy_Estimation.csv")
df.head()
```

Out[ ]:

	Date	Time	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Ligh
0	2017/12/22	10:49:41	24.94	24.75	24.56	25.38	121	34	53	4
1	2017/12/22	10:50:12	24.94	24.75	24.56	25.44	121	33	53	4
2	2017/12/22	10:50:42	25.00	24.75	24.50	25.44	121	34	53	4
3	2017/12/22	10:51:13	25.00	24.75	24.56	25.44	121	34	53	4
4	2017/12/22	10:51:44	25.00	24.75	24.56	25.44	121	34	54	4

In [ ]:
```
df.info()
```

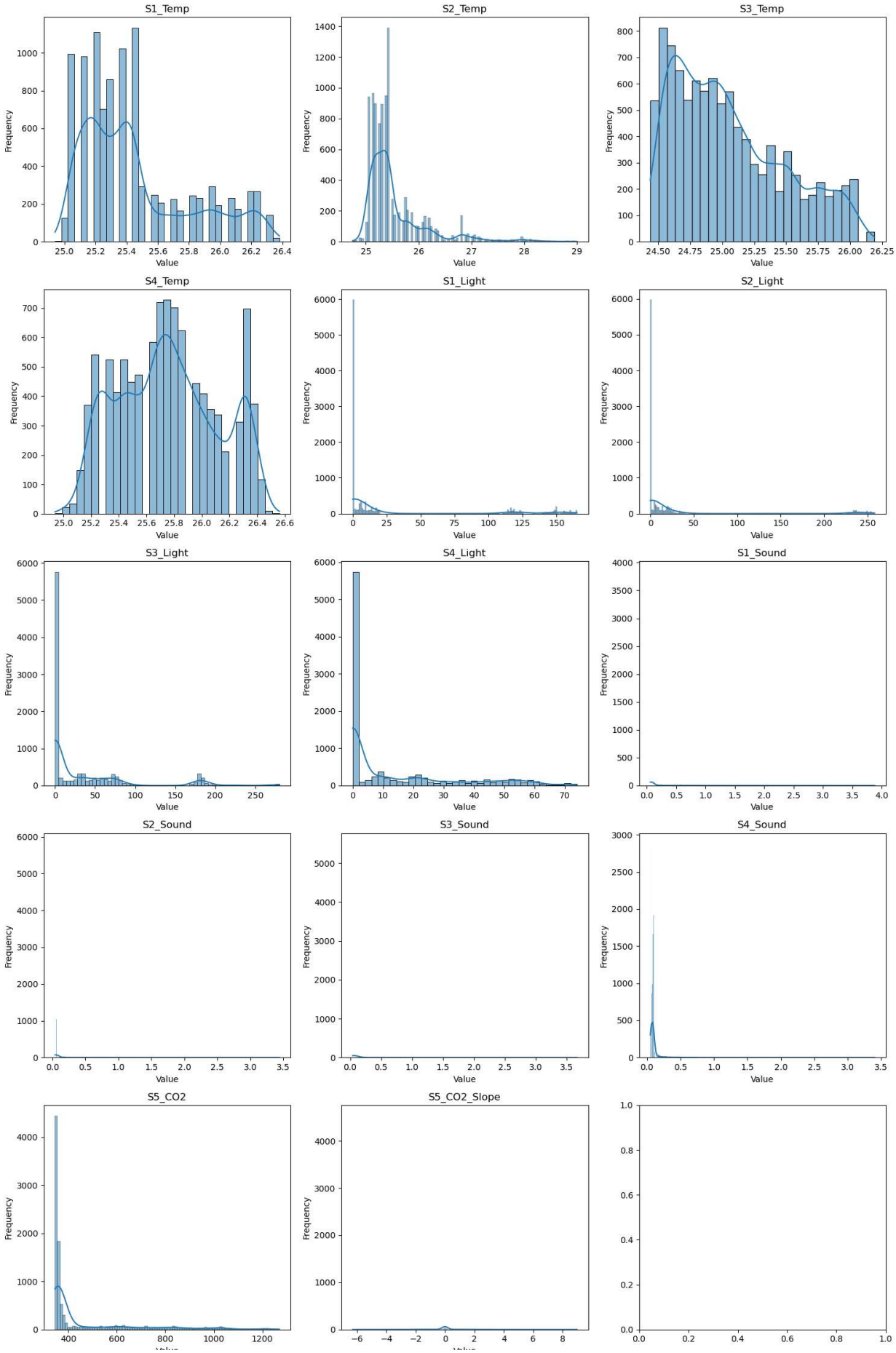
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10129 entries, 0 to 10128
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  10129 non-null  object
 1   Time                  10129 non-null  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
```

In [ ]:
```
categorical_columns = ['S6_PIR', 'S7_PIR', 'Room_Occupancy_Count']
df[categorical_columns] = df[categorical_columns].astype('category')
```

In [ ]:
```
# Extract numeric columns
numeric_columns = ['S1_Temp', 'S2_Temp', 'S3_Temp', 'S4_Temp', 'S1_Light', 'S2_Light',

# Create a grid of histograms
num_plots = len(numeric_columns)
num_cols = 3
num_rows = int(np.ceil(num_plots / num_cols))

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))
fig.suptitle('Histograms of Numeric Columns', fontsize=16)

for i, column in enumerate(numeric_columns):
    row = i // num_cols
    col = i % num_cols
```

```
    ax = axes[row, col]

    # Calculate suitable bin size for the histogram
    bin_size = 10  # Default bin size
    if df[column].max() != df[column].min():
        q1 = df[column].quantile(0.25)
        q3 = df[column].quantile(0.75)
        iqr = q3 - q1
        bin_width = 2 * iqr * (len(df[column]) ** (-1/3))  # Freedman-Diaconis rule
        if bin_width > 0:
            bin_size = int((df[column].max() - df[column].min()) / bin_width)

    sns.histplot(df[column], ax=ax, kde=True, bins=bin_size)
    ax.set_title(column)
    ax.set_xlabel('Value')
    ax.set_ylabel('Frequency')

# Adjust layout
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

# Show the plots
plt.show()
```

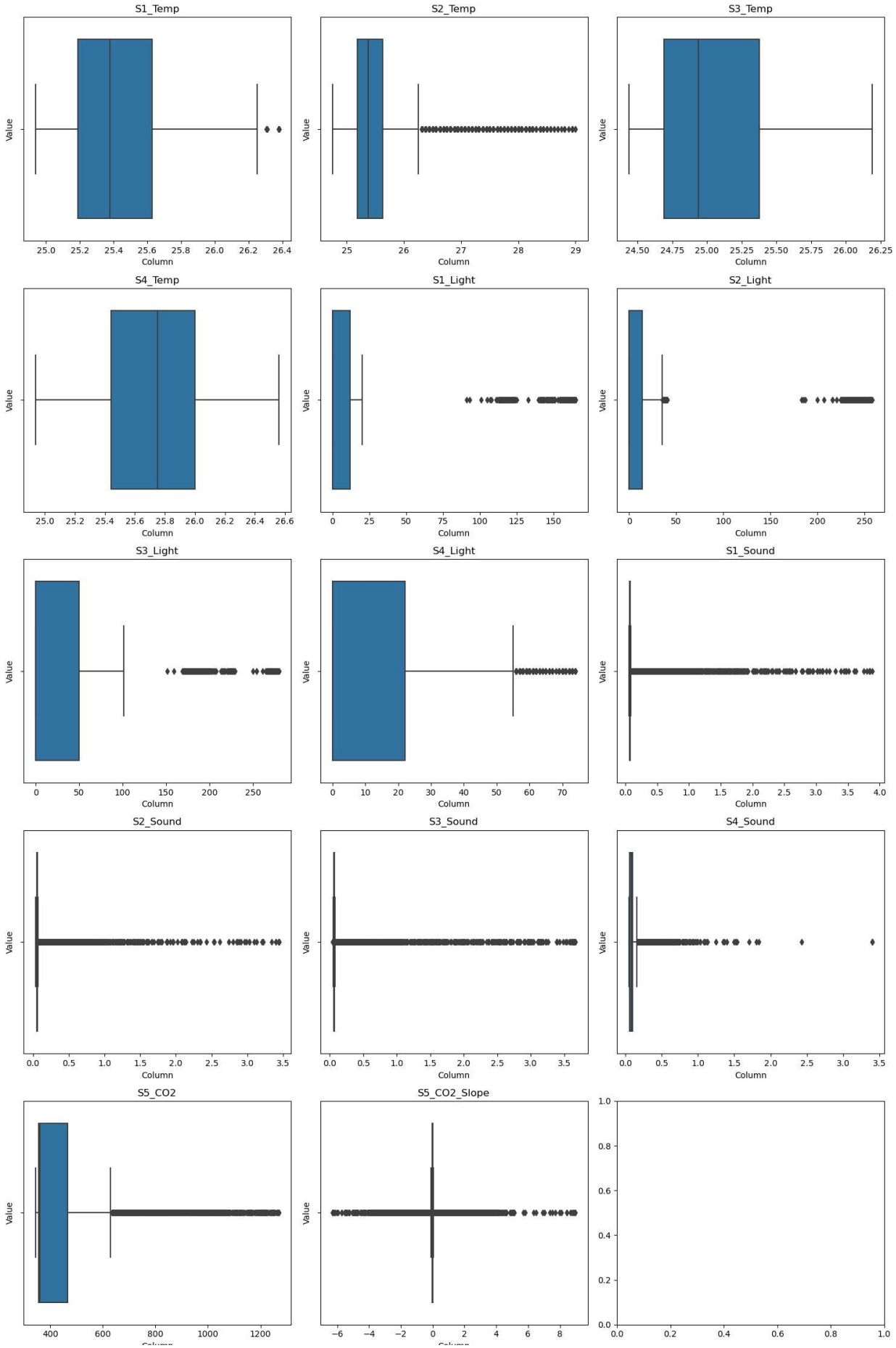
Histograms of Numeric Columns



value value

In [ ]:
```
# Extract numeric columns
numeric_columns = ['S1_Temp', 'S2_Temp', 'S3_Temp', 'S4_Temp', 'S1_Light', 'S2_Light',

# Create a grid of boxplots
num_plots = len(numeric_columns)
num_cols = 3
num_rows = int(np.ceil(num_plots / num_cols))

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))
fig.suptitle('Boxplots of Numeric Columns', fontsize=16)

for i, column in enumerate(numeric_columns):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]

    sns.boxplot(data=df, x=column, ax=ax)
    ax.set_title(column)
    ax.set_xlabel('Column')
    ax.set_ylabel('Value')

# Adjust layout
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

# Show the plots
plt.show()
```

Boxplots of Numeric Columns



Column Column

In [ ]:
```
df['S7_PIR'].value_counts()
```

Out[ ]:
```
0    9323
1     806
Name: S7_PIR, dtype: int64
```

In [ ]:
```
df['S6_PIR'].value_counts()
```

Out[ ]:
```
0    9216
1     913
Name: S6_PIR, dtype: int64
```

In [ ]:
```
df['Room_Occupancy_Count'].value_counts()
```

Out[ ]:
```
0    8228
2     748
3     694
1     459
Name: Room_Occupancy_Count, dtype: int64
```

In [ ]:
```
df= df.dropna()
```

In [ ]:
```
# Extract hour and minute features from the time column
df['Time'] = pd.to_datetime(df['Time'])  # Convert 'Time' column to datetime format
df['Hour'] = df['Time'].dt.hour  # Extract hour
df['Minute'] = df['Time'].dt.minute  # Extract minute
# Drop rows with NaN values
# Drop the original 'Date' and 'Time' columns
df.drop(['Date', 'Time'], axis=1, inplace=True)
df.head()
```

Out[ ]:

	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Light	S1_Sound	S2_Sou
0	24.94	24.75	24.56	25.38	121	34	53	40	0.08	0.
1	24.94	24.75	24.56	25.44	121	33	53	40	0.93	0.
2	25.00	24.75	24.50	25.44	121	34	53	40	0.43	0.
3	25.00	24.75	24.56	25.44	121	34	53	40	0.41	0.
4	25.00	24.75	24.56	25.44	121	34	54	40	0.18	0.

In [ ]:
```
# Drop rows with NaN values
df_cleaned = df.dropna()

# Separate response variable 'y' from the features
y = df_cleaned['Room_Occupancy_Count']
X = df_cleaned.drop(['Room_Occupancy_Count'], axis=1)

# Identify numerical and categorical columns
numerical_columns = ['S1_Temp', 'S2_Temp', 'S3_Temp', 'S4_Temp', 'S1_Light', 'S2_Light
categorical_columns = ['Hour', 'Minute', 'S6_PIR', 'S7_PIR']  # Assuming you extracted

# Handle negative values or zeros before applying log transformation
X[numerical_columns] = np.log1p(np.maximum(X[numerical_columns], 0))
```

```
# One-hot encode categorical columns
X_categorical_encoded = pd.get_dummies(X[categorical_columns], drop_first=True)

# Combine transformed numerical and categorical features
X_transformed_df = pd.concat([X_categorical_encoded, X[numerical_columns]], axis=1)
```

In [ ]:
```
# Split the dataset into train, test, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X_transformed_df, y, test_size=0.2
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_

# Initialize SVM model
svm_model = SVC(kernel="linear", probability=True)

# Fit the SVM model on the training data
svm_model.fit(X_train, y_train)

# Predict on the validation set
y_pred_val = svm_model.predict(X_val)

# Calculate accuracy on the validation set
accuracy = accuracy_score(y_val, y_pred_val)
print(f"Validation Accuracy: {accuracy:.2f}")
```

```
Validation Accuracy: 0.99
```

In [ ]:
```
# Calculate classification report
class_names = [f"Class {i}" for i in range(len(np.unique(y)))]
report = classification_report(y_val, y_pred_val, target_names=class_names)
print("Classification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00       802
     Class 1       1.00      1.00      1.00        49
     Class 2       0.98      0.92      0.95        86
     Class 3       0.89      0.96      0.92        76

    accuracy                           0.99      1013
   macro avg       0.97      0.97      0.97      1013
weighted avg       0.99      0.99      0.99      1013
```

1. **Precision (P):**

   Precision is the ratio of true positive (TP) predictions to the total number of positive predictions (true positives + false positives). It measures the accuracy of the positive predictions made by the model.

   Mathematically: P = TP / (TP + FP)

   Interpretation: Precision answers the question, "Of all instances the model predicted as positive, how many were actually positive?"

2. **Recall (R) or Sensitivity:**

Recall is the ratio of true positive (TP) predictions to the total number of actual positive instances (true positives + false negatives). It measures the ability of the model to capture all actual positive instances.

Mathematically: R = TP / (TP + FN)

Interpretation: Recall answers the question, "Of all actual positive instances, how many did the model correctly predict as positive?"

3. **F1-Score:**

   The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of the model's performance by considering both false positives and false negatives.

   Mathematically: F1 = 2 *(Precision* Recall) / (Precision + Recall)

   Interpretation: The F1-score gives equal weight to precision and recall, making it a useful metric when both false positives and false negatives are important.

The classification report provides an evaluation of the model's performance on each class as well as an overall summary. Here's how to interpret the report:

- **Precision:** Precision measures the accuracy of positive predictions. For a given class, it is the ratio of correctly predicted positive instances to the total instances predicted as positive. Higher precision indicates fewer false positives.
- **Recall (Sensitivity):** Recall measures the ability of the model to capture all positive instances. For a given class, it is the ratio of correctly predicted positive instances to the total actual positive instances. Higher recall indicates fewer false negatives.
- **F1-Score:** The F1-score is the harmonic mean of precision and recall. It provides a balanced view of the model's accuracy, considering both false positives and false negatives. A higher F1-score indicates better overall performance.
- **Support:** Support is the number of actual occurrences of the class in the dataset.
- **Accuracy:** Accuracy is the ratio of correctly predicted instances to the total instances in the dataset.
- **Macro Avg:** The macro average calculates the average precision, recall, and F1-score across all classes. It treats all classes equally regardless of their size.
- **Weighted Avg:** The weighted average calculates the average precision, recall, and F1-score considering the support for each class. It provides a more meaningful average when classes have different sizes.

Interpretation of the provided report:

1. For Class 0, the model has perfect precision, recall, and F1-score of 1.00. This indicates that the model is performing exceptionally well in identifying instances of Class 0.

2. For Class 1, again, the model shows perfect precision, recall, and F1-score of 1.00. It is correctly identifying all instances of Class 1.

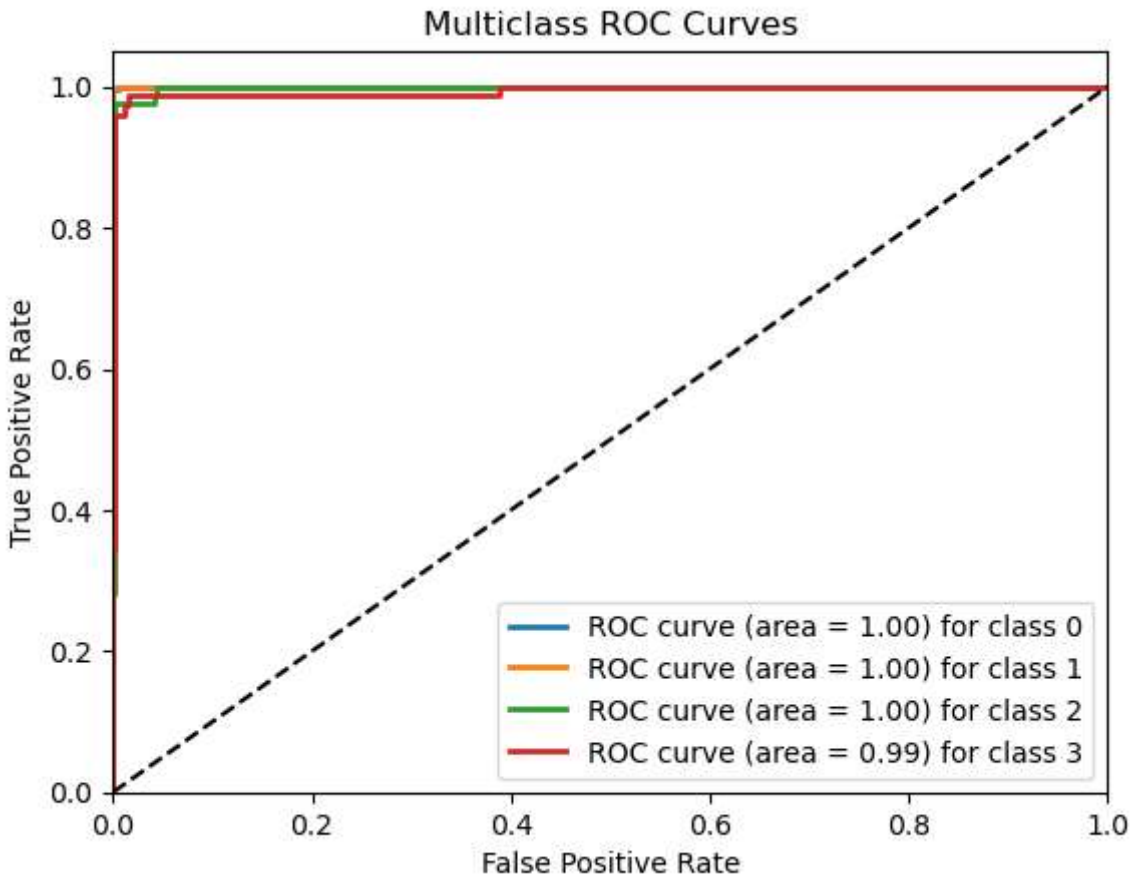
3. For Class 2, the model has a precision of 0.98, indicating that it correctly predicts most positive instances for this class. The recall of 0.92 suggests that some positive instances are not being captured, and the F1-score of 0.95 indicates a good balance between precision and recall.

4. For Class 3, the model has a slightly lower precision of 0.89, meaning that there are some false positive predictions. The recall of 0.96 indicates that it is effectively capturing most positive instances. The F1-score of 0.92 is a good balance between precision and recall.

Overall, the model has achieved high accuracy of 0.99, indicating its strong ability to correctly predict the majority of instances. The macro average and weighted average F1-scores are both around 0.97, indicating good overall model performance. The weighted average considers the class distribution in the dataset, and since Class 0 and Class 1 are larger classes, they contribute more to the weighted average.

In [ ]:
```
# Calculate confusion matrix
conf_matrix = confusion_matrix(y_val, y_pred_val)

# Create an interactive heatmap using plotly with a different color scale
fig = ff.create_annotated_heatmap(conf_matrix, x=class_names, y=class_names,
                                  colorscale='Viridis')  # Change to your preferred co
fig.update_layout(title='Confusion Matrix', xaxis_title='Predicted', yaxis_title='True
fig.show()
```

In [ ]:
```
# Calculate ROC-AUC curves
y_pred_prob_val = svm_model.predict_proba(X_val)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(len(np.unique(y))):
    fpr[i], tpr[i], _ = roc_curve(y_val == i, y_pred_prob_val[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves
plt.figure()
for i in range(len(np.unique(y))):
    plt.plot(fpr[i], tpr[i], lw=2, label='ROC curve (area = %0.2f) for class %d' % (ro

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curves')
plt.legend(loc="lower right")
plt.show()
```



In [ ]:
```
# Perform k-fold cross-validation
cv_scores = cross_val_score(svm_model, X_transformed_df, y, cv=10)
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))
```

```
Cross-Validation Scores: [0.97828233 0.97532083 0.98321816 0.99901283 0.99901283 0.99
901283
 1.         0.97235933 0.9733465  0.92687747]
Mean CV Score: 0.9806443116950005
```

The provided cross-validation scores exemplify the outcomes of a thorough assessment conducted using k-fold cross-validation on the Support Vector Machine (SVM) model. These scores encapsulate the accuracy levels achieved for each fold within the cross-validation procedure. The individual scores range from approximately 92.6% to a perfect 100%. The aggregated metric of primary interest, the mean cross-validation score, stands at around 98.06%. This collective evaluation signifies the SVM model's consistent and commendable performance across diverse data subsets, corroborated by its high average accuracy. The employment of cross-validation contributes to a more comprehensive understanding of the model's generalization capabilities and robustness, bolstering the reliability of the findings beyond a single train-test split.