Notebook Index

1- Import essential libraries

2- Read the data and do explotary data analysis

2.1 Check for duplicates

2.2 Check data types and null values

2.3 Check for classes balance

2.3 result: There is imbalance, we will deal with if the score are affected

2.4 Merge the data with time as DateTime and drop the columns Data and Time

2.5 Check features corralation

2.6 Aggregating features to deal with the corralation issue

2.7 Check the corralation after aggregation

2.8 Pair plot after the aggregation

3- Define and prepare the features and target

3.1 Split the dataset into train and test data

3.2 Scale the features (Standardization)

4- Initiate the machine learning models

4.1 Logistic Regression Model with evaluation metrics and coefficients

4.1 Result: The bench mark model. it has a greate overall scores. but the individual classes (2 and 3) has a low scores

4.2 Gaussian naïve bayes with evaluation metrics and class probabilities

4.2 Result: The second best classifier out of the three classifiers. The score of individual classes are good except Class (3) has a low scores but still acceptable

4.3 XGboost with evaluation metrics and features importance plot

4.3 Result: The highest score out of the three models. Also the score for the indvidual classes are high

1- Import essential libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

2- Read the data and do explotary data analysis

In [2]:

```
df = pd.read_csv('Occupancy_Estimation.csv')
df.head()
```

Out[2]:

	Date	Time	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Lig
0	2017/12/22	10:49:41	24.94	24.75	24.56	25.38	121	34	53	
1	2017/12/22	10:50:12	24.94	24.75	24.56	25.44	121	33	53	
2	2017/12/22	10:50:42	25.00	24.75	24.50	25.44	121	34	53	
3	2017/12/22	10:51:13	25.00	24.75	24.56	25.44	121	34	53	
4	2017/12/22	10:51:44	25.00	24.75	24.56	25.44	121	34	54	

In [3]:

```
df.shape
```

Out[3]:

```
(10129, 19)
```

2.1 Check for duplicates

In [4]:

```
df[df.duplicated()]
```

Out[4]:

Date	Time	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Light	S1_So

In [5]:

```
df.columns
```

Out[5]:

```
Index(['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'],
      dtype='object')
```

2.2 Check data types and null values

In [6]:

```
df.info()
```

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

Loading [MathJax]/extensions/Safe.js

```
 ---  ------                --------------  -----
 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 [7]:

```
df.describe()
```

Out[7]:

	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_L
count	10129.000000	10129.000000	10129.000000	10129.000000	10129.000000	10129.00000	10129.00(
mean	25.454012	25.546059	25.056621	25.754125	25.445059	26.01629	34.248
std	0.351351	0.586325	0.427283	0.356434	51.011264	67.30417	58.40(
min	24.940000	24.750000	24.440000	24.940000	0.000000	0.00000	0.00(
25%	25.190000	25.190000	24.690000	25.440000	0.000000	0.00000	0.00(
50%	25.380000	25.380000	24.940000	25.750000	0.000000	0.00000	0.00(
75%	25.630000	25.630000	25.380000	26.000000	12.000000	14.00000	50.00(
max	26.380000	29.000000	26.190000	26.560000	165.000000	258.00000	280.00(

2.3 Check for classes balance

In [8]:

```
df.Room_Occupancy_Count.value_counts()
```

Out[8]:

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

2.3 result: There is imbalance, we will deal with if the score are affected

2.4 Merge the data with time as DateTime and drop the columns Data and Time

In [9]:

```
DateTimeConc = df['Date'] + ' ' + df['Time']
df['DateTime'] = pd.to_datetime(DateTimeConc)
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10129 entries, 0 to 10128
Data columns (total 20 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
 19  DateTime              10129 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(9), int64(8), object(2)
memory usage: 1.5+ MB
```

In [11]:
```
df.drop(columns=['Date','Time'], inplace = True)
```

In [12]:
```
df.columns
```

Out[12]:
```
Index(['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',
       'DateTime'],
      dtype='object')
```

2.5 Check features corralation

In [13]:
```
df.corr()
```

Out[13]:

	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Li
S1_Temp	1.000000	0.799707	0.948839	0.855279	0.680743	0.548735	0.645163	0.212
S2_Temp	0.799707	1.000000	0.765525	0.696581	0.639773	0.645987	0.607349	0.370
S3_Temp	0.948839	0.765525	1.000000	0.885186	0.594311	0.500054	0.642601	0.301
S4_Temp	0.855279	0.696581	0.885186	1.000000	0.581482	0.456350	0.588459	0.386
S1_Light	0.680743	0.639773	0.594311	0.581482	1.000000	0.842090	0.816438	0.510
S2_Light	0.548735	0.645987	0.500054	0.456350	0.842090	1.000000	0.709579	0.458
S3_Light	0.645163	0.607349	0.642601	0.588459	0.816438	0.709579	1.000000	0.579
S4_Light	0.212217	0.370897	0.301419	0.386871	0.510853	0.458914	0.579484	1.000
S1_Sound	0.436099	0.438274	0.375183	0.355111	0.601166	0.503021	0.502606	0.293
S2_Sound	0.391137	0.409545	0.344026	0.312594	0.534274	0.560630	0.434859	0.303

	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Li
S3_Sound	0.438769	0.427133	0.398177	0.340808	0.494080	0.439269	0.577151	0.169
S4_Sound	0.355826	0.378724	0.326182	0.294939	0.441712	0.413932	0.473606	0.200
S5_CO2	0.866718	0.743722	0.821308	0.650320	0.602740	0.566764	0.650829	0.148
S5_CO2_Slope	0.137391	0.202547	0.095842	0.106208	0.498185	0.493281	0.447708	0.212
S6_PIR	0.436363	0.476901	0.403355	0.340000	0.607159	0.554658	0.501836	0.324
S7_PIR	0.474077	0.465884	0.460309	0.339037	0.545213	0.556797	0.577815	0.220
Room_Occupancy_Count	0.700868	0.671263	0.652047	0.526509	0.849058	0.788764	0.793081	0.355

2.6 Aggregating features to deal with the corralation issue

In [14]:

```
df['S1_4_Temp_Mean'] = (df['S1_Temp'] + df['S2_Temp'] + df['S3_Temp'] + df['S4_Temp']) / 4
df['S1_4_Light_Mean'] = (df['S1_Light'] + df['S2_Light'] + df['S3_Light'] + df['S4_Light'
df['S1_4_Sound_Mean'] = (df['S1_Sound'] + df['S2_Sound'] + df['S3_Sound'] + df['S4_Sound'
df['S6_7_PIR_Sum'] = df['S6_PIR'] + df['S7_PIR']
```

In [15]:

```
#Original_Features = ['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']
Aggregated_Features = ['S1_4_Temp_Mean', 'S1_4_Light_Mean', 'S1_4_Sound_Mean',
       'S6_7_PIR_Sum', 'S5_CO2', 'S5_CO2_Slope']
```

In [16]:

```
list_df_Agg = Aggregated_Features.copy()
list_df_Agg.insert(0, 'Room_Occupancy_Count')
```

In [17]:

```
df[list_df_Agg].describe()
```

Out[17]:

	Room_Occupancy_Count	S1_4_Temp_Mean	S1_4_Light_Mean	S1_4_Sound_Mean	S6_7_PIR_Sur
count	10129.000000	10129.000000	10129.000000	10129.000000	10129.00000
mean	0.398559	25.452704	24.732525	0.137551	0.16971
std	0.893633	0.399201	43.811460	0.231763	0.49376
min	0.000000	24.890000	0.000000	0.052500	0.00000
25%	0.000000	25.127500	0.000000	0.062500	0.00000
50%	0.000000	25.362500	0.000000	0.067500	0.00000
75%	0.000000	25.612500	30.500000	0.075000	0.00000
max	3.000000	26.800000	193.750000	3.182500	2.00000

2.7 Check the corralation after aggregation

In [18]:

```
df[list_df_Agg].corr()
```

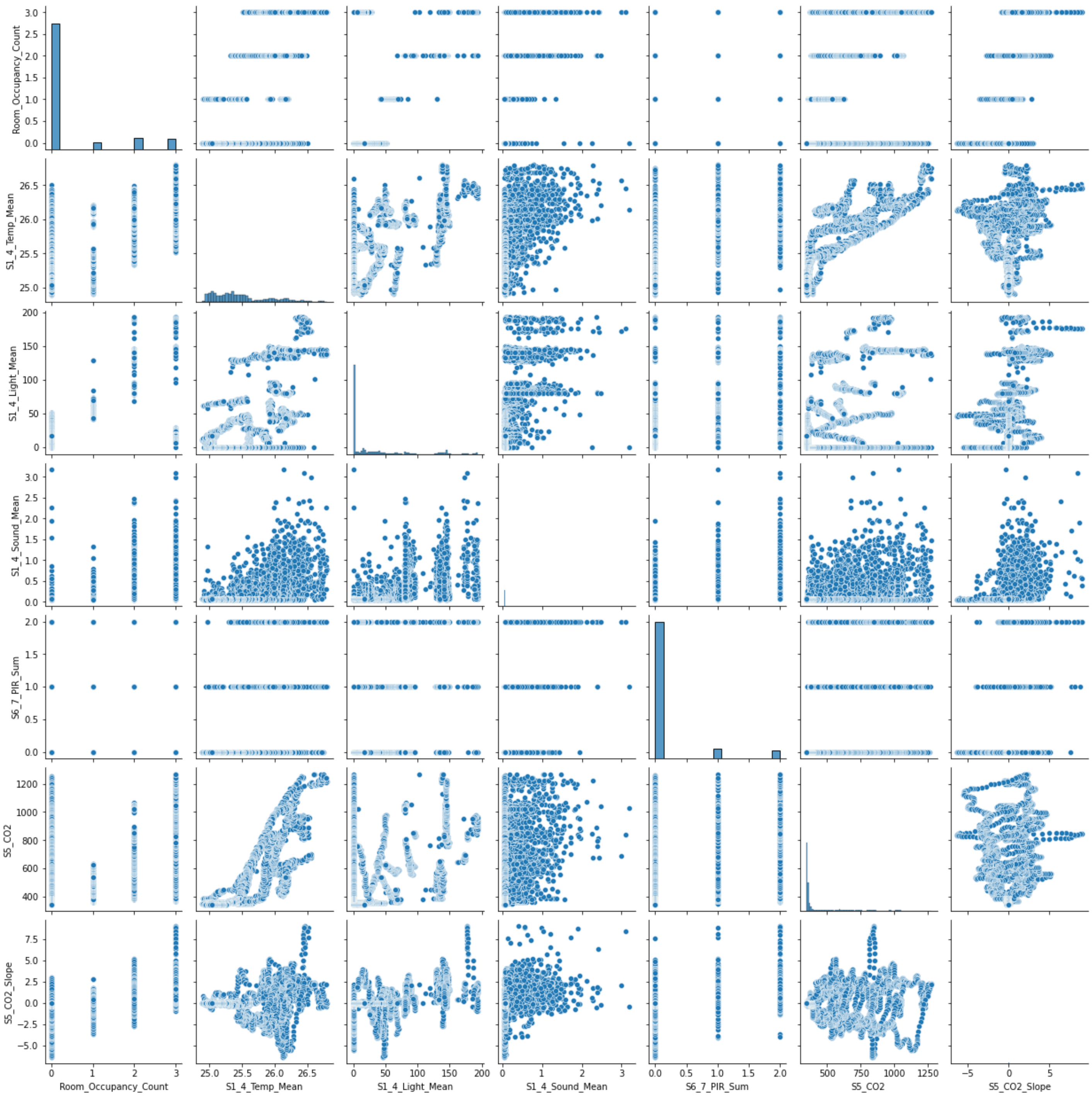
Out[18]:

	Room_Occupancy_Count	S1_4_Temp_Mean	S1_4_Light_Mean	S1_4_Sound_M
Room_Occupancy_Count	1.000000	0.692698	0.854160	0.653
S1_4_Temp_Mean	0.692698	1.000000	0.685873	0.506

	Room_Occupancy_Count	S1_4_Temp_Mean	S1_4_Light_Mean	S1_4_Sound_M
S1_4_Light_Mean	0.854160	0.685873	1.000000	0.646
S1_4_Sound_Mean	0.653527	0.506396	0.646892	1.000
S6_7_PIR_Sum	0.748247	0.523816	0.667364	0.657
S5_CO2	0.660144	0.828725	0.626629	0.472
S5_CO2_Slope	0.601105	0.153957	0.507454	0.401

2.8 Pair plot after the aggregation

In [19]:
```
sns.pairplot(df[list_df_Agg]);
```



3- Define and prepare the features and target

Loading [MathJax]/extensions/Safe.js

3.1 Split the dataset into train and test data

In [20]:

```
from sklearn.model_selection import train_test_split

X = df[Aggregated_Features]
y = df['Room_Occupancy_Count']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

In [21]:

```
print(X_train)
```

```
      S1_4_Temp_Mean  S1_4_Light_Mean  S1_4_Sound_Mean  S6_7_PIR_Sum  S5_CO2  \
1937         25.1425             0.00           0.0600             0     360
4477         25.3300             0.00           0.0600             0     365
8550         25.5475             0.00           0.0625             0     530
1346         25.4850             0.00           0.0600             0     390
7296         25.3475             0.00           0.0700             0     355
...              ...              ...              ...           ...     ...
5734         25.4075            41.50           0.0675             0     355
5191         25.0650            14.25           0.0575             0     360
5390         25.0625            23.50           0.0650             0     355
860          26.3750           144.00           0.4100             1    1160
7270         25.3125             0.00           0.0675             0     355

      S5_CO2_Slope
1937      0.000000
4477      0.000000
8550     -2.592308
1346     -0.642308
7296      0.000000
...            ...
5734      0.000000
5191      0.000000
5390      0.000000
860       3.165385
7270      0.000000

[8103 rows x 6 columns]
```

In [22]:

```
print(X_test)
```

```
      S1_4_Temp_Mean  S1_4_Light_Mean  S1_4_Sound_Mean  S6_7_PIR_Sum  S5_CO2  \
8855         25.2650             0.00           0.0625             0     360
532          25.9675            61.75           0.0650             0     590
1155         25.6725             0.00           0.0600             0     645
7769         25.1125             0.00           0.0725             0     350
4922         25.1275             0.00           0.0575             0     360
...              ...              ...              ...           ...     ...
9396         25.0800             0.00           0.0600             0     345
8359         26.0000            95.25           0.3400             1     830
9609         25.0500             0.00           0.0600             0     345
8936         25.2350             0.00           0.0650             0     350
8513         25.5800             0.00           0.0650             0     620

      S5_CO2_Slope
8855      0.000000
532       0.746154
1155     -2.207692
7769     -0.088462
4922     -0.023077
...
```

Loading [MathJax]/extensions/Safe.js

```
9396     0.000000
8359     0.100000
9609     0.000000
8936    -0.046154
8513    -0.984615

[2026 rows x 6 columns]
```

In [23]:

```python
print(y_train)
```

```
1937    0
4477    0
8550    0
1346    0
7296    0
       ..
5734    0
5191    0
5390    0
860     3
7270    0
Name: Room_Occupancy_Count, Length: 8103, dtype: int64
```

In [24]:

```python
print(y_test)
```

```
8855    0
532     1
1155    0
7769    0
4922    0
       ..
9396    0
8359    2
9609    0
8936    0
8513    0
Name: Room_Occupancy_Count, Length: 2026, dtype: int64
```

3.2 Scale the features (Standardization)

In [25]:

```python
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train_Scl = sc.fit_transform(X_train)
X_test_Scl = sc.transform(X_test)
```

In [26]:

```python
print(X_train_Scl)
```

```
[[-0.76937327 -0.5609514  -0.33326261 -0.34110302 -0.50225845  0.01289966]
 [-0.30095103 -0.5609514  -0.33326261 -0.34110302 -0.47739742  0.01289966]
 [ 0.24241877 -0.5609514  -0.32241377 -0.34110302  0.34301654 -2.20336051]
 ...
 [-0.96923342 -0.01944273 -0.31156494 -0.34110302 -0.52711947  0.01289966]
 [ 2.30972225  2.75722937  1.18557429  1.70627311  3.47550621  2.71910458]
 [-0.34467044 -0.5609514  -0.3007161  -0.34110302 -0.52711947  0.01289966]]
```

In [27]:

```python
print(X_test_Scl)
```

```
[[[illegible] .5609514  -0.32241377 -0.34110302 -0.50225845  0.01289966]
```

```
[ 1.29168458  0.86194903 -0.31156494 -0.34110302  0.64134889  0.65081431]
[ 0.55470026 -0.5609514  -0.33326261 -0.34110302  0.91482021 -1.87453853]
...
[-1.00046157 -0.5609514  -0.33326261 -0.34110302 -0.57684153  0.01289966]
[-0.53828496 -0.5609514  -0.31156494 -0.34110302 -0.5519805  -0.02655898]
[ 0.32361196 -0.5609514  -0.31156494 -0.34110302  0.79051507 -0.82888462]]
```

4- Initiate the machine learning models

4.1 Logistic Regression Model with evaluation metrics and coefficients

In [28]:

```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(random_state = 42, max_iter = 10000)
LR.fit(X_train, y_train)

LR_SclFt = LogisticRegression(random_state = 42, max_iter = 10000)
LR_SclFt.fit(X_train_Scl, y_train)
```

Out[28]:

```
LogisticRegression(max_iter=10000, random_state=42)
```

In [29]:

```
LR_y_pred = LR.predict(X_test)
LR_SclFt_y_pred = LR_SclFt.predict(X_test_Scl)
```

In [41]:

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import cross_val_score, KFold

LR_cm = confusion_matrix(y_test, LR_y_pred)
LR_SclFt_cm = confusion_matrix(y_test, LR_SclFt_y_pred)

print('LR confusion matrix')
print(LR_cm)
print('#######')
print('LR accuracy score')
print(accuracy_score(y_test, LR_y_pred))
print('#######')
print('LR report')
print(classification_report(y_test, LR_y_pred))
print('#######')

cv_X = np.array(X)
cv_y = np.array(y)
kf = KFold(n_splits = 5, shuffle=True, random_state = 42)

print('LR cross validation with f1 weighted average')
LR_cv_results = cross_val_score(LR, cv_X, cv_y, cv=kf, scoring='f1_weighted')
print('Cross validation results is {}'.format(LR_cv_results))
print('Average of cross validation results is {}'.format(np.mean(LR_cv_results)))
print('-------')

print('LR confusion matrix with standardized features')
print(LR_SclFt_cm)
print('#######')
print('LR accuracy score with standardized features')
print(accuracy_score(y_test, LR_SclFt_y_pred))
print('#######')
print('LR report with standardized features')
print(classification_report(y_test, LR_SclFt_y_pred))
```

```
cv_X_scl = sc.fit_transform(cv_X)
print('LR with standardized features cross validation with f1 weighted average')
LR_SclFt_cv_results = cross_val_score(LR_SclFt, cv_X_scl, cv_y, cv=kf, scoring='f1_weight
print('Cross validation results are {}'.format(LR_SclFt_cv_results))
print('Average of cross validation results is {}'.format(np.mean(LR_SclFt_cv_results)))
```

```
LR confusion matrix
[[1613    0    4    2]
 [   9   84   10    0]
 [   0    2  122   40]
 [  10    6   22  102]]
#######
LR accuracy score
0.9481737413622903
#######
LR report
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1619
           1       0.91      0.82      0.86       103
           2       0.77      0.74      0.76       164
           3       0.71      0.73      0.72       140

    accuracy                           0.95      2026
   macro avg       0.85      0.82      0.83      2026
weighted avg       0.95      0.95      0.95      2026

#######
LR cross validation with f1 weighted average
Cross validation results is [0.94774191 0.95089792 0.94831601 0.95264111 0.94800591]
Average of cross validation results is 0.9495205722402724
-------
LR confusion matrix with standardized features
[[1612    1    0    6]
 [  10   83    6    4]
 [   0    3  131   30]
 [   9    7   20  104]]
#######
LR accuracy score with standardized features
0.9526159921026653
#######
LR report with standardized features
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1619
           1       0.88      0.81      0.84       103
           2       0.83      0.80      0.82       164
           3       0.72      0.74      0.73       140

    accuracy                           0.95      2026
   macro avg       0.86      0.84      0.85      2026
weighted avg       0.95      0.95      0.95      2026

#######
LR with standardized features cross validation with f1 weighted average
Cross validation results are [0.95223682 0.9540699  0.95497631 0.95932083 0.9551636 ]
Average of cross validation results is 0.9551534912981046
```

4.1 Result: The bench mark model. it has a greate overall scores. but the individual classes (2 and 3) has a low scores

In [31]:

```
print('#######')
print(LR_SclFt.coef_)
```

```
[[ 4.65979616e-01 -1.66026142e-01 -1.94985926e+00 -3.24425284e+00
  -3.00203303e-03 -2.13288014e+00]
 [ 1.10834027e-01  3.69850666e-02 -7.18874316e-01  2.61240948e-01
  -4.96850398e-03 -2.86291339e-01]
 [-2.08360312e-01  6.68141587e-02  1.57198461e+00  1.16256194e+00
   1.76033769e-03  7.70511130e-01]
 [-3.68453330e-01  6.22269165e-02  1.09674896e+00  1.82044996e+00
   6.21019931e-03  1.64866035e+00]]
#######
[[-1.31974634 -5.93587956 -0.70564651 -1.38611235  0.19838202 -2.2520995 ]
 [ 0.1654534   1.30896293 -0.18427001  0.04224321 -1.22183844 -0.40319526]
 [-0.12698557  2.53291049  0.52014803  0.48863213  0.30915453  0.75540447]
 [ 1.28127851  2.09400614  0.3697685   0.855237    0.71430189  1.89989029]]
```

In [32]:

```
print(LR.intercept_)
print('#######')
print(LR_SclFt.intercept_)
```

```
[ 0.35632858 -0.28565973  0.31666823 -0.38733707]
#######
[ 6.55499954  1.33578078 -1.86414132 -6.026639  ]
```

4.2 Gaussian naïve bayes with evaluation metrics and class probabilities

In [33]:

```
from sklearn.naive_bayes import GaussianNB

GNB = GaussianNB()

GNB.fit(X_train, y_train)
```

Out[33]:

```
GaussianNB()
```

In [34]:

```
GNB_y_pred = GNB.predict(X_test)
```

In [35]:

```
GNB_cm = confusion_matrix(y_test, GNB_y_pred)

print('GNB confusion matrix')
print(GNB_cm)
print('#######')
print('GNB accuracy score')
print(accuracy_score(y_test, GNB_y_pred))
print('#######')
print('GNB report')
print(classification_report(y_test, GNB_y_pred))
print('#######')
print('GNB cross validation with f1 weighted average')
GNB_cv_results = cross_val_score(GNB, cv_X, cv_y, cv=kf, scoring='f1_weighted')
print('Cross validation results are {}'.format(GNB_cv_results))
print('Average of cross validation results is {}'.format(np.mean(GNB_cv_results)))
```

```
GNB confusion matrix
[[1583    1   10   25]
 [   0   97    4    2]
 [   0    0  146   18]
 [   6    6   23  105]]
```

```
GNB accuracy score
0.9531095755182626
#######
GNB report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1619
           1       0.93      0.94      0.94       103
           2       0.80      0.89      0.84       164
           3       0.70      0.75      0.72       140

    accuracy                           0.95      2026
   macro avg       0.86      0.89      0.87      2026
weighted avg       0.96      0.95      0.95      2026

#######
GNB cross validation with f1 weighted average
Cross validation results are [0.95445217 0.96377101 0.963722   0.96339622 0.96838648]
Average of cross validation results is 0.9627455761497805
```

4.2 Result: The second best classifier out of the three classifiers. The score of individual classes are good except Class (3) has a low scores but still acceptable

In [36]:

```python
GNB.theta_
```

Out[36]:

```
array([[ 2.53221993e+01,  7.07198517e+00,  6.80072628e-02,
         3.63141171e-03,  4.04573309e+02, -3.12661057e-01],
       [ 2.57444242e+01,  6.11502809e+01,  2.03771067e-01,
         3.90449438e-01,  4.69747191e+02,  2.85901037e-01],
       [ 2.60079923e+01,  1.09565068e+02,  5.05205479e-01,
         9.00684932e-01,  7.20145548e+02,  1.09521865e+00],
       [ 2.62040027e+01,  1.16901173e+02,  5.26033394e-01,
         1.19314079e+00,  8.55541516e+02,  2.17098723e+00]])
```

4.3 XGboost with evaluation metrics and features importance plot

In [37]:

```python
import xgboost as xgb

xgbM = xgb.XGBClassifier(
                        n_estimators=600,
                        max_depth=4,
                        objective='binary:logistic', #new objective
                        learning_rate=.05,
                        subsample=.8,
                        min_child_weight=3,
                        colsample_bytree=.8,
                        use_label_encoder=False
                       )

eval_set=[(X_train,y_train),(X_test,y_test)]
fit_model = xgbM.fit(
                    X_train, y_train,
                    eval_set=eval_set,
                    eval_metric='mlogloss', #new evaluation metric: classification error
                    early_stopping_rounds=50,
                    verbose=False
                   )
```

In [38]:

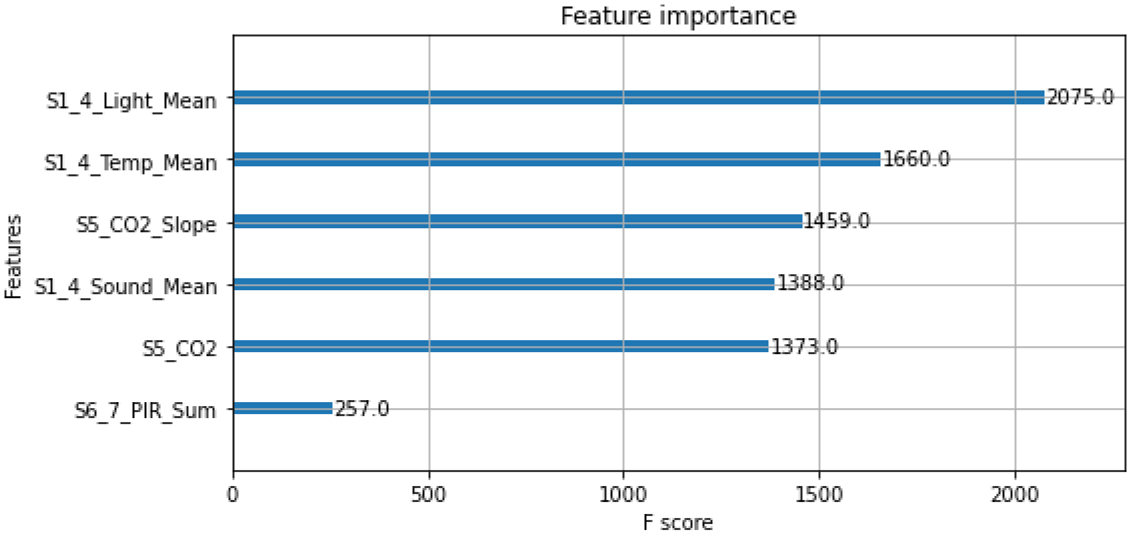
```python
xgbM_y_pred = xgbM.predict(X_test)
```

In [39]:
```
xgbM_cm = confusion_matrix(y_test, xgbM_y_pred)

print('XGB confusion matrix')
print(xgbM_cm)
print('#######')
print('XGB accuracy score')
print(accuracy_score(y_test, xgbM_y_pred))
print('#######')
print('XGB report')
print(classification_report(y_test, xgbM_y_pred))

# to hide warning about mlogloss
xgb.set_config(verbosity=0)
print('#######')
print('XGB cross validation with f1 weighted average')
XGB_cv_results = cross_val_score(xgbM, cv_X, cv_y, cv=kf, scoring='f1_weighted')
print('Cross validation results are {}'.format(XGB_cv_results))
print('Average of cross validation results is {}'.format(np.mean(XGB_cv_results)))
```

```
XGB confusion matrix
[[1618    0    0    1]
 [   0  103    0    0]
 [   0    1  161    2]
 [   2    0    2  136]]
#######
XGB accuracy score
0.9960513326752222
#######
XGB report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1619
           1       0.99      1.00      1.00       103
           2       0.99      0.98      0.98       164
           3       0.98      0.97      0.97       140

    accuracy                           1.00      2026
   macro avg       0.99      0.99      0.99      2026
weighted avg       1.00      1.00      1.00      2026

#######
XGB cross validation with f1 weighted average
Cross validation results are [0.99604279 0.99703742 0.99753129 0.99654521 0.99556268]
Average of cross validation results is 0.9965438770934242
```

4.3 Result: The highest score out of the three models. Also the score for the indvidual classes are high

In [40]:
```
ax = xgb.plot_importance(xgbM)
fig = ax.figure
fig.set_size_inches(8, 4);
```


Feature importance
S1_4_Light_Mean
S1_4_Temp_Mean
S5_CO2_Slope
S1_4_Sound_Mean
S5_CO2
S6_7_PIR_Sum
Features
2075.0
1660.0
1459.0
1388.0
1373.0
257.0
0
500
1000
1500
2000
F score

In [ ]: