```
import numpy as np
import pandas as pd
import scipy.stats
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scipy.stats import chi2_contingency
```

```
from google.colab import files
uploaded= files.upload()
```

```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving EDA_Room_Occupancy.csv to EDA_Room_Occupancy (1).csv
```

```
df= pd.read_csv("EDA_Room_Occupancy.csv")
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     10808 non-null  int64  
 1   Temperature    10808 non-null  float64
 2   Humidity       10808 non-null  float64
 3   Light          10808 non-null  float64
 4   CO2            10808 non-null  float64
 5   HumidityRatio  10808 non-null  float64
 6   Occupancy      10808 non-null  int64  
 7   month          10808 non-null  int64  
 8   day            10808 non-null  int64  
 9   hour           10808 non-null  int64  
 10  minute         10808 non-null  int64  
dtypes: float64(5), int64(6)
memory usage: 928.9 KB
```

```
df= df.drop(['Unnamed: 0'], axis=1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
```

```
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   Occupancy      10808 non-null  int64
 6   month          10808 non-null  int64
 7   day            10808 non-null  int64
 8   hour           10808 non-null  int64
 9   minute         10808 non-null  int64
dtypes: float64(5), int64(5)
memory usage: 844.5 KB
```

```
df.shape
```

```
(10808, 10)
```

```
df.describe()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occup
count	10808.000000	10808.000000	10808.000000	10808.000000	10808.000000	10808.00
mean	20.818940	25.637117	137.494665	617.065584	0.003897	0.24
std	1.075329	4.950347	211.099792	261.214866	0.000785	0.43
min	19.000000	16.745000	0.000000	412.750000	0.002674	0.00
25%	20.000000	21.390000	0.000000	441.000000	0.003323	0.00
50%	20.700000	25.680000	0.000000	464.000000	0.003805	0.00
75%	21.500000	28.324167	413.541667	761.000000	0.004372	0.00
max	23.750000	38.725417	1033.854167	1241.000000	0.005945	1.00

LOGISTIC REGRESSION

```
df1= df[:]
```

```
X = df1.loc[:, df1.columns!= 'Occupancy']
y = df1.loc[:, df1.columns== 'Occupancy']
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
```

```
 5   month          10808 non-null  int64
 6   day            10808 non-null  int64
 7   hour           10808 non-null  int64
 8   minute         10808 non-null  int64
dtypes: float64(5), int64(4)
memory usage: 760.1 KB
```

```
y
```

	Occupancy
0	1
1	1
2	1
3	1
4	1
...	...
10803	1
10804	1
10805	1
10806	1
10807	1

10808 rows × 1 columns

```
''' Splitting the data into Train & Test (70-30 respectively) '''

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)


train = X_train.join(y_train)


train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7565 entries, 5519 to 2732
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Temperature    7565 non-null   float64
 1   Humidity       7565 non-null   float64
 2   Light          7565 non-null   float64
 3   CO2            7565 non-null   float64
 4   HumidityRatio  7565 non-null   float64
```

```
 5   month          7565 non-null   int64
 6   day            7565 non-null   int64
 7   hour           7565 non-null   int64
 8   minute         7565 non-null   int64
 9   Occupancy      7565 non-null   int64
dtypes: float64(5), int64(5)
memory usage: 970.1 KB
```

```
no_Occupancy = train[train.Occupancy == 0]
len(no_Occupancy)
```

```
5718
```

```
yes_Occupancy = train[train.Occupancy == 1]
len(yes_Occupancy)
```

```
1847
```

```
from sklearn.utils import resample


# Smote is done - over sampling
yes_Occupancy_os = resample(yes_Occupancy,
                            replace = True,
                            n_samples = len(no_Occupancy),
                            random_state = 14)


train_os = pd.concat([yes_Occupancy_os, no_Occupancy])


train_os.Occupancy.value_counts()
```

```
1    5718
0    5718
Name: Occupancy, dtype: int64
```

```
X_train_os = train_os.loc[:, train_os.columns != 'Occupancy']
y_train_os = train_os.loc[:, train_os.columns == 'Occupancy']
```

#Recurrsive Feature Elimination

```
from sklearn import datasets
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression


logreg = LogisticRegression(max_iter=10000000)


rfe = RFE(logreg, n_features_to_select=6)
```

```
rfe = rfe.fit(X_train_os, y_train_os.values.ravel())
```

```
rfe.n_features_to_select
```

```
6
```

```
X_train_os.columns[rfe.get_support()]
```

```
Index(['Temperature', 'Humidity', 'Light', 'month', 'day', 'minute'], dtype='object')
```

```
cols = X_train_os.columns[rfe.get_support()]
```

```
cols.to_list()
```

```
['Temperature', 'Humidity', 'Light', 'month', 'day', 'minute']
```

#Logistic Model by statistic apporach

```
#_____________________________________sm model to see p_values
x1 = X_train_os[cols]
```

```
x1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11436 entries, 2601 to 2732
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  11436 non-null  float64
 1   Humidity     11436 non-null  float64
 2   Light        11436 non-null  float64
 3   month        11436 non-null  int64  
 4   day          11436 non-null  int64  
 5   minute       11436 non-null  int64  
dtypes: float64(3), int64(3)
memory usage: 625.4 KB
```

```
y_train_os.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11436 entries, 2601 to 2732
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Occupancy  11436 non-null  int64
dtypes: int64(1)
memory usage: 178.7 KB
```

```
y_train_os.value_counts()
```

```
Occupancy
1            5718
0            5718
dtype: int64
```

```
y1 = y_train_os
```

```
y1
```

	Occupancy
2601	1
2556	1
5484	1
9707	1
5333	1
...	...
7891	0
9225	0
4859	0
3264	0
2732	0

11436 rows × 1 columns

```
#_____________Stats model
import statsmodels.api as sm
```

```
x1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11436 entries, 2601 to 2732
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  11436 non-null  float64
 1   Humidity     11436 non-null  float64
 2   Light        11436 non-null  float64
 3   month        11436 non-null  int64  
 4   day          11436 non-null  int64  
 5   minute       11436 non-null  int64  
dtypes: float64(3), int64(3)
memory usage: 625.4 KB
```

```
y1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11436 entries, 2601 to 2732
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Occupancy  11436 non-null  int64
dtypes: int64(1)
memory usage: 178.7 KB
```

```
logit_model = sm.Logit(y1,x1)
```

```
result = logit_model.fit(method='bfgs')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/discrete/discrete_model.py:1736: R
  return 1/(1+np.exp(-X))
/usr/local/lib/python3.7/dist-packages/statsmodels/discrete/discrete_model.py:1789: R
  return np.sum(np.log(self.cdf(q*np.dot(X,params))))
/usr/local/lib/python3.7/dist-packages/statsmodels/discrete/discrete_model.py:1736: R
  return 1/(1+np.exp(-X))
/usr/local/lib/python3.7/dist-packages/statsmodels/discrete/discrete_model.py:1789: R
  return np.sum(np.log(self.cdf(q*np.dot(X,params))))
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.057369
         Iterations: 35
         Function evaluations: 45
         Gradient evaluations: 40
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarn
  "Check mle_retvals", ConvergenceWarning)
```

```
print(result.summary2())
```

```
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.917
Dependent Variable: Occupancy        AIC:              1324.1428
Date:               2021-12-15 10:16 BIC:              1368.2099
No. Observations:   11436            Log-Likelihood:   -656.07
Df Model:           5                LL-Null:          -7926.8
Df Residuals:       11430            LLR p-value:      0.0000
Converged:          0.0000           Scale:            1.0000
-----------------------------------------------------------------
                Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
-----------------------------------------------------------------
Temperature    -0.5383   0.0337 -15.9693 0.0000 -0.6043 -0.4722
Humidity        0.2309   0.0206  11.2104 0.0000  0.1906  0.2713
Light           0.0257   0.0009  28.2780 0.0000  0.0240  0.0275
month          -0.0981   0.0512  -1.9169 0.0553 -0.1984  0.0022
day            -0.3191   0.0437  -7.3038 0.0000 -0.4047 -0.2334
minute         -0.0131   0.0046  -2.8535 0.0043 -0.0221 -0.0041
=================================================================
```

#Logistic model by SK learn method

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

```
logreg= LogisticRegression(solver= 'sag')
logreg.fit(x1, y1)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:985: DataConversic
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:354: ConvergenceW
  ConvergenceWarning,
LogisticRegression(solver='sag')
```

```
## X_test should alsso have only 2 columns
X_test2= X_test[cols]
```

```
X_test2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3243 entries, 5862 to 3981
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  3243 non-null   float64
 1   Humidity     3243 non-null   float64
 2   Light        3243 non-null   float64
 3   month        3243 non-null   int64  
 4   day          3243 non-null   int64  
 5   minute       3243 non-null   int64  
dtypes: float64(3), int64(3)
memory usage: 177.4 KB
```

```
y_pred= logreg.predict(X_test2)
```

```
log_score= logreg.score(X_test2, y_test)
```

```
print("Accuracy of logistic regression classifier on test data:{}".format(log_score))
```

```
Accuracy of logistic regression classifier on test data:0.9821153253160654
```

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[2333   56]
 [   2  852]]
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.98      0.99      2389
               1       0.94      1.00      0.97       854

        accuracy                           0.98      3243
       macro avg       0.97      0.99      0.98      3243
    weighted avg       0.98      0.98      0.98      3243
```

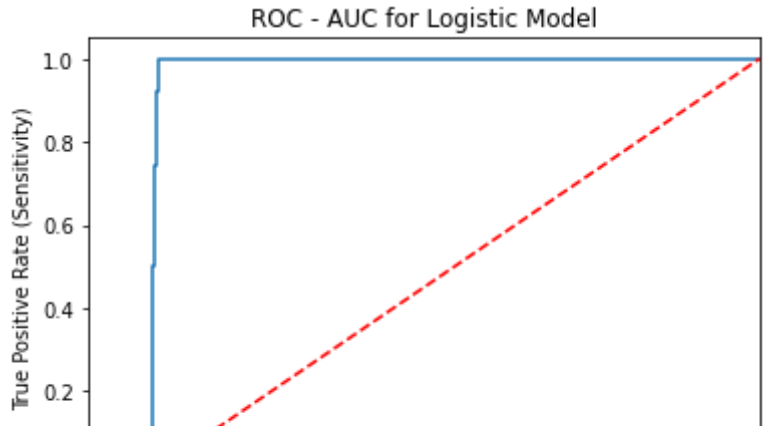
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve


logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test2))
logit_roc_auc
```

```
    0.9871086547142788
```

```
""  Area under curve is 0.987   ""
```

```
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test2)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.1, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1-specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('ROC - AUC for Logistic Model')
plt.legend(loc="lower right")
plt.show()
```



 # DECISION TREE

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   Occupancy      10808 non-null  int64  
 6   month          10808 non-null  int64  
 7   day            10808 non-null  int64  
 8   hour           10808 non-null  int64  
 9   minute         10808 non-null  int64  
dtypes: float64(5), int64(5)
memory usage: 844.5 KB
```

```
X_train_os = train_os.loc[:, train_os.columns != 'Occupancy']
y_train_os = train_os.loc[:, train_os.columns == 'Occupancy']
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   month          10808 non-null  int64  
 6   day            10808 non-null  int64  
 7   hour           10808 non-null  int64  
 8   minute         10808 non-null  int64  
dtypes: float64(5), int64(4)
memory usage: 760.1 KB
```

```
y
```

	Occupancy
0	1
1	1
2	1
3	1
4	1

```
'''Fit Tree'''
#train test - split

from sklearn.model_selection import train_test_split
```

10805	1

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

10807	1

```
from sklearn.tree import DecisionTreeClassifier


#fit tree on train data
#model
clf = DecisionTreeClassifier()


#Fit Classifier model on train set
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
#Predict/estimate_train  X_train
y_pred_train = clf.predict(X_train)


y_pred_train
```

```
array([1, 0, 0, ..., 1, 1, 0])
```

```
#Predict/estimate_test  X_test
y_pred_test = clf.predict(X_test)


y_pred_test
```

```
array([0, 0, 0, ..., 1, 0, 1])
```

```
#See the train
from sklearn import tree


tree.plot_tree(clf.fit(X_train, y_train))
```

```
[Text(98.3426674364896, 211.4, 'X[2] <= 365.125\ngini = 0.369\nsamples = 7565\nvalue
 Text(46.392609699769054, 199.32, 'X[1] <= 37.718\ngini = 0.004\nsamples = 5623\nvalu
 Text(40.20692840646651, 187.24, 'X[2] <= 289.25\ngini = 0.003\nsamples = 5621\nvalue
 Text(24.74272517321016, 175.16, 'X[3] <= 1238.833\ngini = 0.001\nsamples = 5510\nval
 Text(12.37136258660508, 163.07999999999998, 'X[2] <= 212.0\ngini = 0.0\nsamples = 54
 Text(6.18568129330254, 151.0, 'gini = 0.0\nsamples = 5391\nvalue = [5391, 0]'),
 Text(18.55704387990762, 151.0, 'X[2] <= 213.875\ngini = 0.027\nsamples = 73\nvalue =
 Text(12.37136258660508, 138.92000000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]
 Text(24.74272517321016, 138.92000000000002, 'gini = 0.0\nsamples = 72\nvalue = [72,
 Text(37.11408775981524, 163.07999999999998, 'X[1] <= 33.256\ngini = 0.043\nsamples =
 Text(30.9284064665127, 151.0, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(43.299769053117785, 151.0, 'gini = 0.0\nsamples = 45\nvalue = [45, 0]'),
 Text(55.67113163972286, 175.16, 'X[5] <= 6.0\ngini = 0.102\nsamples = 111\nvalue = [
 Text(49.48545034642032, 163.07999999999998, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]
 Text(61.8568129330254, 163.07999999999998, 'X[5] <= 8.5\ngini = 0.019\nsamples = 106
 Text(55.67113163972286, 151.0, 'gini = 0.0\nsamples = 105\nvalue = [105, 0]'),
 Text(68.04249422632795, 151.0, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(52.57829099307159, 187.24, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(150.29272517321016, 199.32, 'X[3] <= 472.833\ngini = 0.102\nsamples = 1942\nval
 Text(86.59953810623557, 187.24, 'X[8] <= 49.5\ngini = 0.496\nsamples = 22\nvalue = [
 Text(80.41385681293302, 175.16, 'X[3] <= 447.05\ngini = 0.444\nsamples = 18\nvalue =
 Text(74.22817551963048, 163.07999999999998, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]
 Text(86.59953810623557, 163.07999999999998, 'X[4] <= 0.003\ngini = 0.32\nsamples = 1
 Text(80.41385681293302, 151.0, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]'),
 Text(92.78521939953811, 151.0, 'X[6] <= 3.0\ngini = 0.375\nsamples = 4\nvalue = [1,
 Text(86.59953810623557, 138.92000000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]
 Text(98.97090069284064, 138.92000000000002, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]
 Text(92.78521939953811, 175.16, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(213.98591224018475, 187.24, 'X[4] <= 0.005\ngini = 0.092\nsamples = 1920\nvalue
 Text(173.58568129330254, 175.16, 'X[0] <= 22.264\ngini = 0.147\nsamples = 1155\nvalu
 Text(111.34226327944572, 163.07999999999998, 'X[0] <= 19.55\ngini = 0.064\nsamples =
 Text(105.15658198614318, 151.0, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(117.52794457274827, 151.0, 'X[2] <= 602.125\ngini = 0.06\nsamples = 875\nvalue
 Text(111.34226327944572, 138.92000000000002, 'X[3] <= 816.25\ngini = 0.058\nsamples
 Text(85.82632794457274, 126.84, 'X[0] <= 22.223\ngini = 0.102\nsamples = 443\nvalue
 Text(79.6406466512702, 114.75999999999999, 'X[3] <= 815.875\ngini = 0.099\nsamples =
 Text(73.45496535796767, 102.67999999999999, 'X[8] <= 31.5\ngini = 0.095\nsamples = 4
 Text(34.79445727482679, 90.6, 'X[4] <= 0.004\ngini = 0.018\nsamples = 221\nvalue = [
 Text(28.60877598152425, 78.52000000000001, 'gini = 0.0\nsamples = 137\nvalue = [0, 1
 Text(40.98013856812933, 78.52000000000001, 'X[1] <= 23.945\ngini = 0.046\nsamples =
 Text(28.60877598152425, 66.44, 'X[3] <= 650.75\ngini = 0.5\nsamples = 2\nvalue = [1,
 Text(22.42309468822171, 54.359999999999985, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]
 Text(34.79445727482679, 54.359999999999985, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]
 Text(53.35150115473441, 66.44, 'X[6] <= 2.0\ngini = 0.024\nsamples = 82\nvalue = [1,
 Text(47.16581986143187, 54.359999999999985, 'X[8] <= 10.5\ngini = 0.105\nsamples = 1
 Text(40.98013856812933, 42.28, 'X[3] <= 746.167\ngini = 0.278\nsamples = 6\nvalue =
 Text(34.79445727482679, 30.19999999999999, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]
 Text(47.16581986143187, 30.19999999999999, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]
 Text(53.35150115473441, 42.28, 'gini = 0.0\nsamples = 12\nvalue = [0, 12]'),
 Text(59.53718244803695, 54.359999999999985, 'gini = 0.0\nsamples = 64\nvalue = [0, 6
 Text(112.11547344110855, 90.6, 'X[1] <= 25.268\ngini = 0.165\nsamples = 220\nvalue =
 Text(96.65127020785219, 78.52000000000001, 'X[3] <= 804.125\ngini = 0.11\nsamples =
 Text(84.2799076212471, 66.44, 'X[3] <= 653.5\ngini = 0.088\nsamples = 195\nvalue = [
 Text(78.09422632794457, 54.359999999999985, 'X[3] <= 648.75\ngini = 0.195\nsamples =
 Text(71.90854503464203, 42.28, 'X[4] <= 0.004\ngini = 0.14\nsamples = 79\nvalue = [6
 Text(59.53718244803695, 30.19999999999999, 'X[3] <= 646.5\ngini = 0.029\nsamples = 6
 Text(53.35150115473441, 18.120000000000005, 'gini = 0.0\nsamples = 64\nvalue = [0, 6
```

```
#Model has learnt unneccesaary things
#Need to optimize
```

```
from sklearn.metrics import accuracy_score
```

```
Text(90.46558891454966, 54.359999999999985, 'gini = 0.0\nsamples = 113\nvalue = [0,
```

```
print(round(accuracy_score(y_train,y_pred_train), 2))
```

```
1.0
```

```
Text(115.20831408775982, 54.359999999999985, 'gini = 0.0\nsamples = 7\nvalue = [0, 7
```

```
print(round(accuracy_score(y_test,y_pred_test), 2))
```

```
0.99
```

```
Text(92.01200923787529, 114.75999999999999, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]
```

```
# Accuracy of train data is 1.0
# Accuracy of test data is 0.99
```

```
Text(130.67251732101616, 102.67999999999999, 'X[3] <= 1039.25\ngini = 0.021\nsamples
```

```
from sklearn import tree
```

```
Text(149.2295612009238, 114.75999999999999, 'X[4] <= 0.005\ngini = 0.245\nsamples =
```

```
path = clf.cost_complexity_pruning_path(X_train, y_train)
```

```
Text(123.7136258660508, 138.92000000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]
```

```
path
```

```
{'ccp_alphas': array([0.00000000e+00, 8.70504409e-05, 8.76048094e-05, 1.18968936e-04,
        1.22019421e-04, 1.30214756e-04, 1.31574307e-04, 1.32139726e-04,
        1.76250275e-04, 1.94676441e-04, 1.98281560e-04, 1.98281560e-04,
        1.98281560e-04, 2.48526821e-04, 2.54207128e-04, 2.61881305e-04,
        2.64029824e-04, 2.68745749e-04, 2.86406698e-04, 3.78537523e-04,
        3.90462149e-04, 4.11250643e-04, 4.14588716e-04, 4.19196841e-04,
        4.81540931e-04, 4.95490255e-04, 5.75750900e-04, 6.15867402e-04,
        6.16204081e-04, 7.41353990e-04, 1.07405358e-03, 1.30731210e-03,
        1.71280287e-03, 3.40184690e-01]),
 'impurities': array([0.00000000e+00, 2.61151323e-04, 5.23965751e-04, 7.61903623e-04,
        1.00594247e-03, 1.26637198e-03, 1.79266920e-03, 2.32122810e-03,
        2.49747838e-03, 2.69215482e-03, 2.89043638e-03, 3.08871794e-03,
        3.28699950e-03, 4.03257996e-03, 4.28678709e-03, 4.54866840e-03,
        4.81269822e-03, 5.35018972e-03, 5.63659642e-03, 6.01513394e-03,
        6.40559609e-03, 6.81684673e-03, 8.06061288e-03, 1.30909750e-02,
        1.35725159e-02, 1.40680062e-02, 1.46437571e-02, 1.52596245e-02,
        1.71082367e-02, 1.85909447e-02, 2.07390518e-02, 2.20463639e-02,
        2.88975754e-02, 3.69082265e-01])}
```

```
Text(248.20046189376444, 90.6, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')
```

```
alphas = path['ccp_alphas']
```

```
Text(285.31454965357966, 114.75999999999999, 'X[2] <= 639.667\ngini = 0.135\nsamples
```

```
alphas
```

```
array([0.00000000e+00, 8.70504409e-05, 8.76048094e-05, 1.18968936e-04,
       1.22019421e-04, 1.30214756e-04, 1.31574307e-04, 1.32139726e-04,
       1.76250275e-04, 1.94676441e-04, 1.98281560e-04, 1.98281560e-04,
       1.98281560e-04, 2.48526821e-04, 2.54207128e-04, 2.61881305e-04,
       2.64029824e-04, 2.68745749e-04, 2.86406698e-04, 3.78537523e-04,
```

```
            3.90462149e-04, 4.11250643e-04, 4.14588716e-04, 4.19196841e-04,
            4.81540931e-04, 4.95490255e-04, 5.75750900e-04, 6.15867402e-04,
            6.16204081e-04, 7.41353990e-04, 1.07405358e-03, 1.30731210e-03,
            1.71280287e-03, 3.40184690e-01])
```

```
acrcy_train, acrcy_test = [],[]

for i in alphas:
    clf = DecisionTreeClassifier(ccp_alpha=i)

    clf.fit(X_train, y_train)
    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)

    acrcy_train.append(accuracy_score(y_train, y_pred_train))
    acrcy_test.append(accuracy_score(y_test,y_pred_test))
```

```
acrcy_train
```

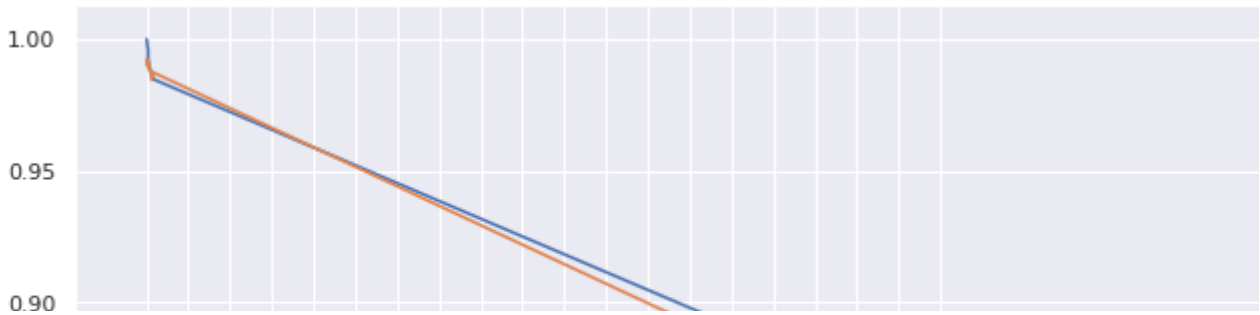
```
[1.0,
 0.9998678122934567,
 0.9997356245869135,
 0.9996034368803701,
 0.9994712491738268,
 0.9993390614672836,
 0.9990746860541969,
 0.9988103106411104,
 0.998678122934567,
 0.9985459352280238,
 0.9981493721083939,
 0.9981493721083939,
 0.9981493721083939,
 0.9977528089887641,
 0.9976206212822207,
 0.9974884335756775,
 0.9973562458691342,
 0.9970918704560476,
 0.9969596827495043,
 0.9966953073364178,
 0.9964309319233311,
 0.9961665565102445,
 0.9949768671513549,
 0.9924653007270324,
 0.9922009253139458,
 0.9919365499008592,
 0.9912756113681428,
 0.9911434236615995,
 0.9902181097157964,
 0.9892927957699934,
 0.9881031064111038,
 0.9865168539325843,
 0.9847984137475215,
 0.7558493060145407]
```

```
acrcy_test
```

```
[0.9919827320382362,
 0.9910576626580326,
 0.9913660191181005,
 0.9907493061979649,
 0.9913660191181005,
 0.9907493061979649,
 0.9916743755781684,
 0.992291088498304,
 0.992291088498304,
 0.9919827320382362,
 0.9919827320382362,
 0.9916743755781684,
 0.9919827320382362,
 0.9916743755781684,
 0.9913660191181005,
 0.9913660191181005,
 0.9910576626580326,
 0.9910576626580326,
 0.9910576626580326,
 0.9910576626580326,
 0.9901325932778292,
 0.9901325932778292,
 0.9898242368177613,
 0.990440949737897,
 0.9898242368177613,
 0.9907493061979649,
 0.9892075238976257,
 0.9892075238976257,
 0.9892075238976257,
 0.9879740980573543,
 0.9876657415972865,
 0.9845821769966081,
 0.9873573851372186,
 0.736663583102066]
```

```
# now we have scores
# lets, plot

sns.set()
plt.figure(figsize = (14,7))
sns.lineplot(y =acrcy_train, x = alphas, label = 'Train_Accuracy')
sns.lineplot(y =acrcy_test, x = alphas, label = 'Test_Accuracy')
plt.xticks(ticks=np.arange(0.00,0.2,0.01))
plt.xlabel('alphas')
plt.ylabel('accuracy')
plt.show()
```



```
#_________with ccp = 0.04
clf = DecisionTreeClassifier(ccp_alpha=0.04, random_state = 14)


clf.fit(X_train,y_train)
```

```
     DecisionTreeClassifier(ccp_alpha=0.04, random_state=14)
```

```
y_pred_train = clf.predict(X_train)


y_pred_test = clf.predict(X_test)


from sklearn.metrics import accuracy_score
print(round(accuracy_score(y_train,y_pred_train), 2))
```

```
     0.98
```

```
print(round(accuracy_score(y_test,y_pred_test), 2))
```

```
     0.99
```

```
### Confusion Matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


confusion_matrix = confusion_matrix(y_test, y_pred_test)
print(confusion_matrix)
```

```
     [[2350   39]
      [   2  852]]
```

```
### Classification Report
from sklearn.metrics import classification_report


print(classification_report(y_test, y_pred_test))
```

```
                   precision    recall  f1-score   support

               0       1.00      0.98      0.99      2389
               1       0.96      1.00      0.98       854

        accuracy                           0.99      3243
       macro avg       0.98      0.99      0.98      3243
    weighted avg       0.99      0.99      0.99      3243
```

```
############################  ROC AUC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_curve, auc, roc_auc_score


predictedProbability = clf.predict_proba(X_test)[:, 1]
fpr,tpr, thresholds = metrics.roc_curve(y_test, predictedProbability)


fpr
```

```
    array([0.        , 0.01632482, 1.        ])
```

```
tpr
```

```
    array([0.        , 0.99765808, 1.        ])
```

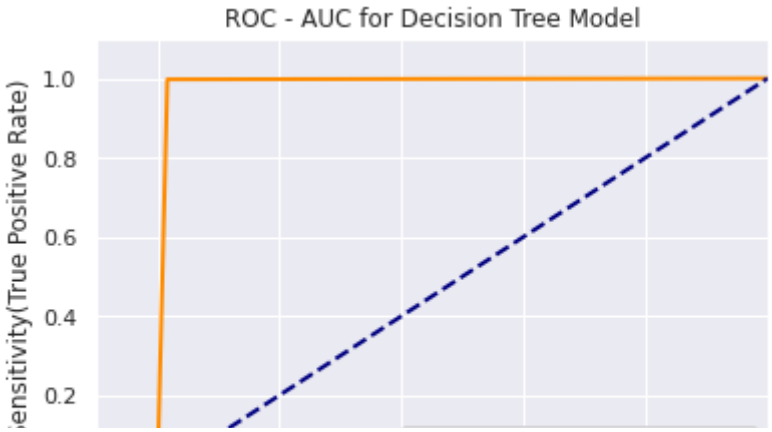
```
thresholds
```

```
    array([1.94593203e+00, 9.45932029e-01, 1.77841010e-03])
```

```
dff = pd.DataFrame(dict(fpr = fpr,tpr = tpr))
auc = auc(fpr,tpr)
auc
```

```
    0.9906666287619976
```

```
plt.figure()
lw = 2
plt.plot(fpr, tpr, color = 'darkorange',
         lw =lw, label = 'ROC Curve (area = %0.2f)' %auc)
plt.plot([0,1],[0,1], color='navy', lw = lw, linestyle = '--')
plt.xlim([-0.1, 1.0])
plt.ylim([0.0, 1.1])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title("ROC - AUC for Decision Tree Model")
plt.legend(loc = "lower right")
plt.show()
```



RANDOM FOREST

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   Occupancy      10808 non-null  int64  
 6   month          10808 non-null  int64  
 7   day            10808 non-null  int64  
 8   hour           10808 non-null  int64  
 9   minute         10808 non-null  int64  
dtypes: float64(5), int64(5)
memory usage: 844.5 KB
```

```
X_train_os = train_os.loc[:, train_os.columns != 'Occupancy']
y_train_os = train_os.loc[:, train_os.columns == 'Occupancy']
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   month          10808 non-null  int64  
 6   day            10808 non-null  int64  
 7   hour           10808 non-null  int64  
```

```
 8   minute         10808 non-null  int64
dtypes: float64(5), int64(4)
memory usage: 760.1 KB
```

y

	Occupancy
0	1
1	1
2	1
3	1
4	1
...	...
10803	1
10804	1
10805	1
10806	1
10807	1

10808 rows × 1 columns

```
'''Fit Tree'''
#train test - split

from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)


#import the classifier
from sklearn.ensemble import RandomForestClassifier


#Create Classifier object
#in our previous experiment, we found ccp_alphas = 0.04 has the best accuarcy
clf_rf = RandomForestClassifier(n_estimators =100, ccp_alpha= 0.04, random_state = 14)


#fit the classifier with x and y  data = train
mod_rf = clf_rf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning
```

```
#Prediction
y_train_pred = mod_rf.predict(X_train)
y_train_pred
```

```
array([1, 0, 0, ..., 1, 1, 0])
```

```
#Prediction
y_test_pred = mod_rf.predict(X_test)
y_test_pred
```

```
array([0, 0, 0, ..., 1, 0, 1])
```

```
from sklearn.metrics import accuracy_score

print(round(accuracy_score(y_train,y_train_pred), 2))
```

```
0.98
```

```
print(round(accuracy_score(y_test,y_test_pred), 2))
```

```
0.98
```

```
#_________________________________ Extract Feature Importance
fi = pd.DataFrame({'feature': list(X_train.columns),
                   'importance': mod_rf.feature_importances_}).\
    sort_values('importance', ascending = False)
```

```
fi.head()
```

	feature	importance
2	Light	0.511094
3	CO2	0.304064
0	Temperature	0.089617
7	hour	0.057141
4	HumidityRatio	0.027836

```
# Accuarcy 2 cells above is 0.98 & 0.98 for Train & test (respectively)
# This accuracy is for having all columns as features in our model
# Lets build a model keeping 2 best features
# that is keeping 'Light' , 'CO2' only
from sklearn.ensemble import RandomForestClassifier
```

```
#Create Classifier object
#in our previous experiment Decision Tree model,
#we found ccp_alphas = 0.035 has the best accuarcy
clf_rf1 = RandomForestClassifier(n_estimators =100, ccp_alpha= 0.04, random_state = 14)
```

```
# fit the classifier with x and y data=TRAIN,
#this time with Failure_Type only
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7565 entries, 5519 to 2732
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    7565 non-null   float64
 1   Humidity       7565 non-null   float64
 2   Light          7565 non-null   float64
 3   CO2            7565 non-null   float64
 4   HumidityRatio  7565 non-null   float64
 5   month          7565 non-null   int64  
 6   day            7565 non-null   int64  
 7   hour           7565 non-null   int64  
 8   minute         7565 non-null   int64  
dtypes: float64(5), int64(4)
memory usage: 591.0 KB
```

```
X_train1 = X_train.iloc[ : ,[2,3]]
X_train1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7565 entries, 5519 to 2732
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Light   7565 non-null   float64
 1   CO2     7565 non-null   float64
dtypes: float64(2)
memory usage: 177.3 KB
```

```
mod_rf1 = clf_rf1.fit(X_train1, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DataConversionWarning
  """Entry point for launching an IPython kernel.
```

```
#Prediction
y_train_pred1 = mod_rf1.predict(X_train1)
y_train_pred1
```

```
array([1, 0, 0, ..., 1, 1, 0])
```

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3243 entries, 5862 to 3981
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    3243 non-null   float64
```

```
 1   Humidity       3243 non-null   float64
 2   Light          3243 non-null   float64
 3   CO2            3243 non-null   float64
 4   HumidityRatio  3243 non-null   float64
 5   month          3243 non-null   int64
 6   day            3243 non-null   int64
 7   hour           3243 non-null   int64
 8   minute         3243 non-null   int64
dtypes: float64(5), int64(4)
memory usage: 253.4 KB
```

```
X_test1 = X_test.iloc[ : ,[2,3]]
X_test1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3243 entries, 5862 to 3981
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Light   3243 non-null   float64
 1   CO2     3243 non-null   float64
dtypes: float64(2)
memory usage: 76.0 KB
```

```
#Prediction
y_test_pred1 = mod_rf1.predict(X_test1)
y_test_pred1
```

```
array([0, 0, 0, ..., 1, 0, 1])
```

```
from sklearn.metrics import accuracy_score
```

```
print(round(accuracy_score(y_train,y_train_pred1), 2))
```

```
0.98
```

```
print(round(accuracy_score(y_test,y_test_pred1), 2))
```

```
0.99
```

```
### There is no much difference in accuarcy
#Earlier train accuracy = 0.98 now with 2 features its 0.98
#Earlier test accuracy = 0.98 now with 2 features its 0.98
```

```
### Confusion Matrix
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix = confusion_matrix(y_test, y_test_pred1)
print(confusion_matrix)
```

```
[[2349   40]
```

```
       [   5  849]]
```

```
### Classification Report
print(classification_report(y_test, y_test_pred1))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.98      0.99      2389
               1       0.96      0.99      0.97       854

        accuracy                           0.99      3243
       macro avg       0.98      0.99      0.98      3243
    weighted avg       0.99      0.99      0.99      3243
```

```
############################  ROC AUC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_curve, auc, roc_auc_score


predictedProbability1 = mod_rf1.predict_proba(X_test1)[:, 1]
fpr,tpr, thresholds = metrics.roc_curve(y_test, predictedProbability1)


fpr
```

```
    array([0.00000000e+00, 4.18585182e-04, 1.17203851e-02, 1.21389703e-02,
           1.21389703e-02, 1.21389703e-02, 1.21389703e-02, 1.25575555e-02,
           1.25575555e-02, 1.25575555e-02, 1.29761406e-02, 1.29761406e-02,
           1.29761406e-02, 1.29761406e-02, 1.29761406e-02, 1.29761406e-02,
           1.29761406e-02, 1.63248221e-02, 1.67434073e-02, 1.67434073e-02,
           1.67434073e-02, 7.07408958e-02, 7.86940142e-02, 7.99497698e-02,
           8.37170364e-02, 9.20887401e-02, 9.54374215e-02, 1.02134784e-01,
           1.03809125e-01, 1.04646296e-01, 1.09669318e-01, 1.12599414e-01,
           1.14692340e-01, 1.15529510e-01, 1.17203851e-01, 1.18459607e-01,
           1.19296777e-01, 1.20133947e-01, 1.23064044e-01, 1.00000000e+00])
```

```
tpr
```

```
    array([0.        , 0.01639344, 0.85362998, 0.85831382, 0.8618267 ,
           0.87002342, 0.88173302, 0.8969555 , 0.90046838, 0.90163934,
           0.91100703, 0.91569087, 0.92271663, 0.92505855, 0.92857143,
           0.93091335, 0.93325527, 0.9941452 , 0.9941452 , 0.99765808,
           0.99882904, 0.99882904, 0.99882904, 0.99882904, 0.99882904,
           0.99882904, 0.99882904, 0.99882904, 0.99882904, 0.99882904,
           0.99882904, 0.99882904, 0.99882904, 0.99882904, 1.        ])
```

```
thresholds
```

```
    array([1.88230225, 0.88230225, 0.8814641 , 0.87484168, 0.87408866,
           0.86628095, 0.8655035 , 0.85796363, 0.85055406, 0.84318438,
           0.83853817, 0.82930016, 0.82196696, 0.71072826, 0.6441427 ,
           0.63684772, 0.52635246, 0.50996269, 0.50362834, 0.48223706,
```

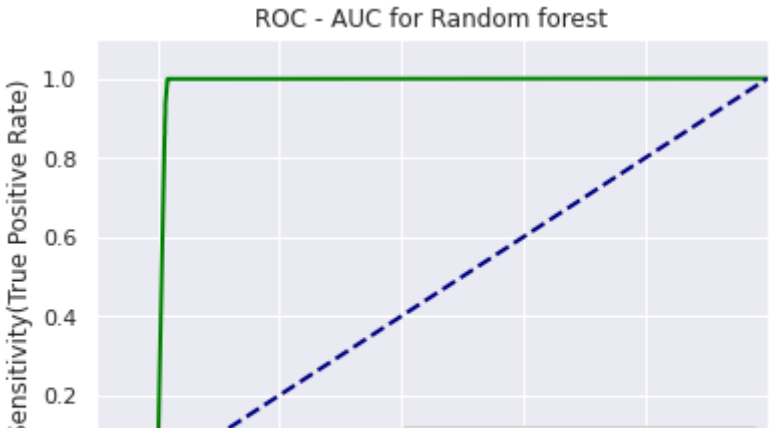
```
              0.40743401, 0.27606965, 0.26777633, 0.26011641, 0.25196408,
              0.24428268, 0.24428079, 0.2367428 , 0.22933324, 0.19969356,
              0.19942138, 0.19236036, 0.09237772, 0.08565025, 0.08508274,
              0.05023504, 0.02462451, 0.02221881, 0.02202243, 0.01513416])
```

```
dff1 = pd.DataFrame(dict(fpr = fpr,tpr = tpr))
auc1 = auc(fpr,tpr)
auc1
```

```
     0.9922917097587204
```

```
''' Area Under Curve is 0.99 '''
```

```
plt.figure()
lw = 2
plt.plot(fpr, tpr, color = 'green',
         lw =lw, label = 'ROC Curve (area = %0.2f)' %auc1)
plt.plot([0,1],[0,1], color='navy', lw = lw, linestyle = '--')
plt.xlim([-0.1, 1.0])
plt.ylim([0.0, 1.1])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title("ROC - AUC for Random forest")
plt.legend(loc = "lower right")
plt.show()
```



GRADIENT BOOSTING MODEL

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
```

```
df.info()
```

```
     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 10808 entries, 0 to 10807
```

```
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   Occupancy      10808 non-null  int64  
 6   month          10808 non-null  int64  
 7   day            10808 non-null  int64  
 8   hour           10808 non-null  int64  
 9   minute         10808 non-null  int64  
dtypes: float64(5), int64(5)
memory usage: 844.5 KB
```

```
X_train_os = train_os.loc[:, train_os.columns != 'Occupancy']
y_train_os = train_os.loc[:, train_os.columns == 'Occupancy']
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   month          10808 non-null  int64  
 6   day            10808 non-null  int64  
 7   hour           10808 non-null  int64  
 8   minute         10808 non-null  int64  
dtypes: float64(5), int64(4)
memory usage: 760.1 KB
```

```
y
```

	Occupancy
0	1
1	1
2	1

```
'''Fit Tree'''
#train test - split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

GB = GradientBoostingClassifier()
```

10806	1

```
GB_mod = GB.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494: DataConversionWar
  y = column_or_1d(y, warn=True)
```

```
# Prediction
y_train_GB = GB_mod.predict(X_train)
y_train_GB
```

```
array([1, 0, 0, ..., 1, 1, 0])
```

```
# Prediction
y_test_GB = GB_mod.predict(X_test)
y_test_GB
```

```
array([0, 0, 0, ..., 1, 0, 1])
```

```
print(round(accuracy_score(y_train, y_train_GB), 2))
```

```
0.99
```

```
print(round(accuracy_score(y_test, y_test_GB), 2))
```

```
0.99
```

```
### Confusion Matrix
from sklearn.metrics import confusion_matrix


confusion_matrix = confusion_matrix(y_test, y_test_GB)
print(confusion_matrix)
```

```
[[2368   21]
 [   6  848]]
```

```
### Classification Report
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_test_GB))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.99      0.99      2389
               1       0.98      0.99      0.98       854

        accuracy                           0.99      3243
       macro avg       0.99      0.99      0.99      3243
    weighted avg       0.99      0.99      0.99      3243
```

KNOWING YOUR NEAREST NEIGHBORS

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
 4   HumidityRatio  10808 non-null  float64
 5   Occupancy      10808 non-null  int64  
 6   month          10808 non-null  int64  
 7   day            10808 non-null  int64  
 8   hour           10808 non-null  int64  
 9   minute         10808 non-null  int64  
dtypes: float64(5), int64(5)
memory usage: 844.5 KB
```

```
X_train_os = train_os.loc[:, train_os.columns != 'Occupancy']
y_train_os = train_os.loc[:, train_os.columns == 'Occupancy']
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10808 entries, 0 to 10807
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Temperature    10808 non-null  float64
 1   Humidity       10808 non-null  float64
 2   Light          10808 non-null  float64
 3   CO2            10808 non-null  float64
```

```
 4   HumidityRatio  10808 non-null  float64
 5   month          10808 non-null  int64
 6   day            10808 non-null  int64
 7   hour           10808 non-null  int64
 8   minute         10808 non-null  int64
dtypes: float64(5), int64(4)
memory usage: 760.1 KB
```

```
y
```

	Occupancy
0	1
1	1
2	1
3	1
4	1
...	...
10803	1
10804	1
10805	1
10806	1
10807	1

10808 rows × 1 columns

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)


from sklearn.neighbors import KNeighborsClassifier


#Building Model @ n_neighbors = 6

knn = KNeighborsClassifier(n_neighbors = 6)
print(knn)
mpm_knn = knn.fit(X_train, y_train)
print(mpm_knn)
```

```
KNeighborsClassifier(n_neighbors=6)
KNeighborsClassifier(n_neighbors=6)
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198: Data
  return self._fit(X, y)
```

```
#Applying on Test data for prediction
y_pred_KNN = mpm_knn.predict(X_test)
print(y_pred_KNN)
```

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

```
#Prediction Score
mpm_knn.score(X_test, y_test)
```

```
0.9870490286771508
```

```
#Accuracy Score
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred_KNN)
```

```
0.9870490286771508
```

```
# creating a confusion matrix
from sklearn.metrics import confusion_matrix
knn_predictions = knn.predict(X_test)
cm = confusion_matrix(y_test, knn_predictions)
cm
```

```
array([[2367,   22],
       [  20,  834]])
```

```
### Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test, knn_predictions))
```

```
                  precision    recall  f1-score   support

               0       0.99      0.99      0.99      2389
               1       0.97      0.98      0.98       854

        accuracy                           0.99      3243
       macro avg       0.98      0.98      0.98      3243
    weighted avg       0.99      0.99      0.99      3243
```