In [1]:
```
#import libraries
import numpy as np
import pandas as pd
import keras

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, precision_score, recall_score

from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam
```

In [2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
#read the train data
train_data=pd.read_csv(r'C:\Users\96891\OneDrive\Documents\sonia\smoke_detection_iot.csv')
```

In [4]:
```
#print the data
train_data.head()
```

Out[4]:

	Unnamed: 0	UTC	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	CNT	F Ala
0	0	1654733331	20.000	57.36	0	400	12306	18520	939.735	0.0	0.0	0.0	0.0	0.0	0	
1	1	1654733332	20.015	56.67	0	400	12345	18651	939.744	0.0	0.0	0.0	0.0	0.0	1	
2	2	1654733333	20.029	55.96	0	400	12374	18764	939.738	0.0	0.0	0.0	0.0	0.0	2	
3	3	1654733334	20.044	55.28	0	400	12390	18849	939.736	0.0	0.0	0.0	0.0	0.0	3	
4	4	1654733335	20.059	54.69	0	400	12403	18921	939.744	0.0	0.0	0.0	0.0	0.0	4	

In [5]:
```
#dropping unnecessary columns
train_data.drop(['Unnamed: 0', 'UTC', 'CNT'], axis=1, inplace=True)
```

In [6]:
```
#check updated data
train_data.head()
```

Out[6]:

	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	Fire Alarm
0	20.000	57.36	0	400	12306	18520	939.735	0.0	0.0	0.0	0.0	0.0	0
1	20.015	56.67	0	400	12345	18651	939.744	0.0	0.0	0.0	0.0	0.0	0
2	20.029	55.96	0	400	12374	18764	939.738	0.0	0.0	0.0	0.0	0.0	0
3	20.044	55.28	0	400	12390	18849	939.736	0.0	0.0	0.0	0.0	0.0	0
4	20.059	54.69	0	400	12403	18921	939.744	0.0	0.0	0.0	0.0	0.0	0

In [7]:
```
#data pre-proccessing
#splitting the dependent and independent variable
x=train_data.drop('Fire Alarm', axis=1)
y=train_data['Fire Alarm']
```

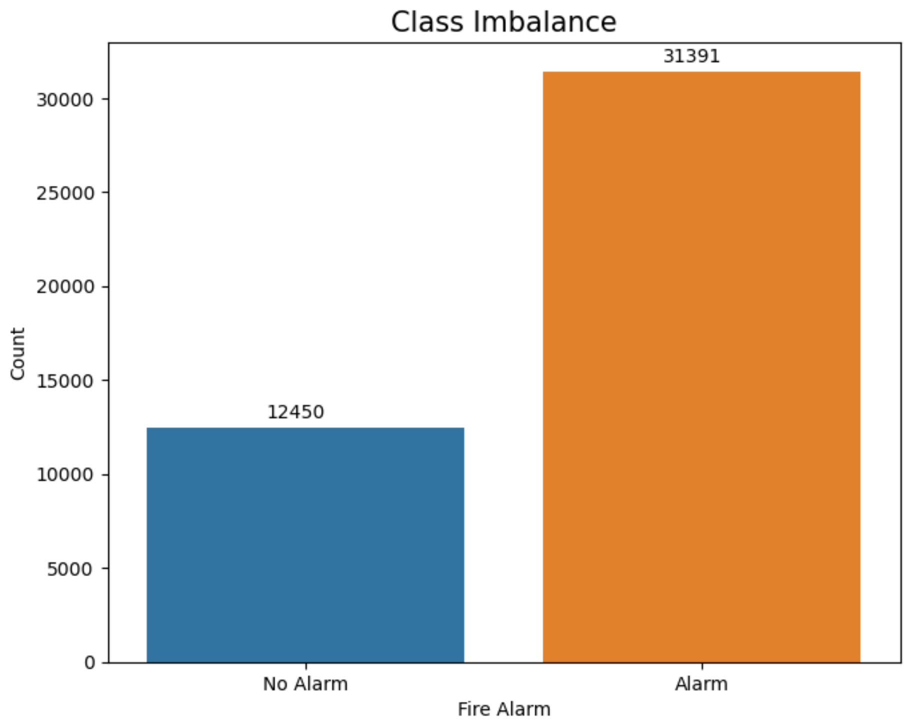
In [8]:
```
#splitting the data into training and testing sets
X_train, X_test, Y_train, Y_test= train_test_split(x,y,test_size=0.3,random_state=0)
#random_state=0, we get the same train and test sets accross different executions
```

In [9]:
```
#print the dimensions of the train and test data
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(43841, 12)
(43841,)
(18789, 12)
(18789,)
```

In [10]:
```
# the scale of each feature is very different, so we need to bring all of them to the same scale.
ss= StandardScaler()
X_train=ss.fit_transform(X_train)
X_test= ss.transform(X_test)
```

In [11]:
```
#class distribution
#check if the target classes are balanced
sns.countplot(x = Y_train)
plt.text(x = 0 - 0.1, y = Y_train.value_counts()[0] + 500, s = Y_train.value_counts()[0])
plt.text(x = 1 - 0.1, y = Y_train.value_counts()[1] + 500, s = Y_train.value_counts()[1])
plt.xticks([0, 1], ['No Alarm', 'Alarm'])
plt.ylabel('Count')
plt.tight_layout(pad = -1)
plt.title('Class Imbalance', fontsize = 15)
plt.show()
```



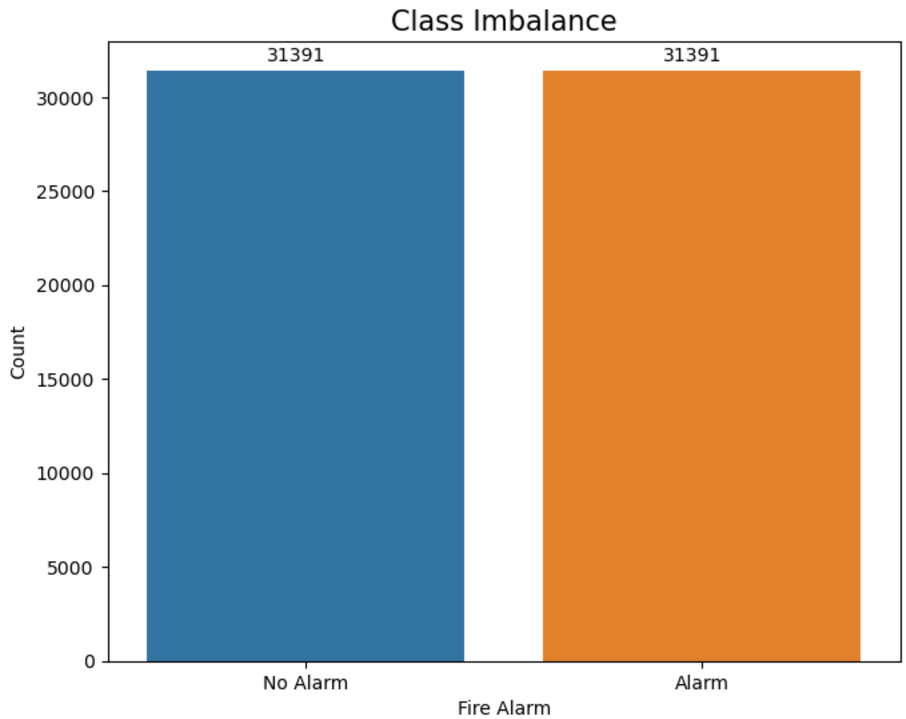
In [12]:
```
pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\96891\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\96891\anaconda3\lib\site-packages (from imblearn) (0.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\96891\anaconda3\lib\site-packages (from imbalanced-learn->imblea
rn) (1.0.2)
Requirement already satisfied: scipy>=1.3.2 in c:\users\96891\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.
9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\96891\anaconda3\lib\site-packages (from imbalanced-learn->imble
arn) (2.2.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\96891\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.2.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\96891\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

In [13]:
```
#data is highly biased, will result in a biased model
#solution:Synthetic Minority Over-sampling Technique
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state = 10)
X_train, Y_train = smote.fit_resample(X_train, Y_train)
```

In [14]:
```
#check classes again
sns.countplot(x = Y_train)
plt.text(x = 0 - 0.1, y = Y_train.value_counts()[0] + 500, s = Y_train.value_counts()[0])
plt.text(x = 1 - 0.1, y = Y_train.value_counts()[1] + 500, s = Y_train.value_counts()[1])
plt.xticks([0, 1], ['No Alarm', 'Alarm'])
plt.ylabel('Count')
plt.tight_layout(pad = -1)
plt.title('Class Imbalance', fontsize = 15)
plt.show()
```



In [15]:
```
#now that the data is balanced, we can build the model
#Dense Neural Network
#Model Architecture
model=Sequential([
    Dense(units=32, activation='relu',input_shape=(12,),name="Layer1"),
    Dense(units=64,activation='relu',name="Layer2"),
    Dense(units=128, activation='relu', name="Layer3"),
    Dense(units=1, activation='sigmoid', name="Output")
])
#relu activation function is used in the hidden layers and sigmoid activation function is used in the output layer
```

In [16]:
```
#before training, we must compile the model
model.compile(loss='binary_crossentropy', optimizer= 'Adam', metrics=['accuracy'])
```

In [17]:
```
#fit the model
model.fit(X_train, Y_train, validation_split=0.1, batch_size=10, epochs=10, shuffle=True, verbose=2)
```

```
Epoch 1/10
5651/5651 - 17s - loss: 0.0576 - accuracy: 0.9770 - val_loss: 0.0113 - val_accuracy: 0.9962 - 17s/epoch - 3ms/step
Epoch 2/10
5651/5651 - 16s - loss: 0.0239 - accuracy: 0.9910 - val_loss: 0.0131 - val_accuracy: 0.9947 - 16s/epoch - 3ms/step
Epoch 3/10
5651/5651 - 16s - loss: 0.0206 - accuracy: 0.9929 - val_loss: 0.0402 - val_accuracy: 0.9853 - 16s/epoch - 3ms/step
Epoch 4/10
5651/5651 - 15s - loss: 0.0161 - accuracy: 0.9938 - val_loss: 0.0127 - val_accuracy: 0.9952 - 15s/epoch - 3ms/step
Epoch 5/10
5651/5651 - 16s - loss: 0.0143 - accuracy: 0.9945 - val_loss: 0.0169 - val_accuracy: 0.9927 - 16s/epoch - 3ms/step
Epoch 6/10
5651/5651 - 16s - loss: 0.0126 - accuracy: 0.9955 - val_loss: 0.0070 - val_accuracy: 0.9976 - 16s/epoch - 3ms/step
Epoch 7/10
5651/5651 - 16s - loss: 0.0119 - accuracy: 0.9959 - val_loss: 0.0100 - val_accuracy: 0.9963 - 16s/epoch - 3ms/step
Epoch 8/10
5651/5651 - 15s - loss: 0.0099 - accuracy: 0.9962 - val_loss: 0.0061 - val_accuracy: 0.9973 - 15s/epoch - 3ms/step
Epoch 9/10
5651/5651 - 15s - loss: 0.0129 - accuracy: 0.9963 - val_loss: 0.0077 - val_accuracy: 0.9971 - 15s/epoch - 3ms/step
Epoch 10/10
5651/5651 - 15s - loss: 0.0121 - accuracy: 0.9967 - val_loss: 0.0068 - val_accuracy: 0.9970 - 15s/epoch - 3ms/step
```

Out[17]:
```
<keras.callbacks.History at 0x2a7a478fd60>
```

In [18]:
```
#evaluate the model on testing data
model.evaluate(X_test,Y_test)
```

```
588/588 [==============================] - 1s 2ms/step - loss: 0.0058 - accuracy: 0.9983
```

Out[18]:
```
[0.005776534788310528, 0.998296856880188]
```

In [19]:
```
Y_true,Y_pred=Y_test, np.round(model.predict(X_test))
```

```
588/588 [==============================] - 1s 2ms/step
```

In [20]:
```
#calculate
f1=f1_score(Y_true, Y_pred)
acc=accuracy_score(Y_true, Y_pred)
precision=precision_score(Y_true, Y_pred)
recall=recall_score(Y_true, Y_pred)
cm=confusion_matrix(Y_true, Y_pred)
```

In [21]:
```
#print
print(f"F1 Score : {f1}\n")
print(f"Accuracy : {acc}\n")
print(f"Precision : {precision}\n")
print(f"Recall : {recall}\n")
print(f"Confusion Matrix : {cm}\n")
```

```
F1 Score : 0.9988029328146042

Accuracy : 0.9982968758316036

Precision : 0.9988029328146042

Recall : 0.9988029328146042

Confusion Matrix : [[ 5407    16]
 [   16 13350]]
```

In [22]:
```
TN=cm[0][0]
FN=cm[1][0]
FP=cm[0][1]
TP=cm[1][1]
```

In [23]:
```
print ("True Positive= ", TP)
print ("True Negative= ", TN)
print ("False Positive= ", FP)
print ("False Negative= ", FN)
```

```
True Positive=  13350
True Negative=  5407
False Positive=  16
False Negative=  16
```

In [24]:
```
#specificity
print ("Specifity=", TN/(TN+FP))
```

```
Specifity= 0.9970496035404758
```

In [25]:
```
#sensitivity
print ("Sensitivity=", TP/(TP+FN))
```

```
Sensitivity= 0.9988029328146042
```

In [26]:
```
#print classification report
from sklearn.metrics import classification_report
print (classification_report (Y_true, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5423
           1       1.00      1.00      1.00     13366

    accuracy                           1.00     18789
   macro avg       1.00      1.00      1.00     18789
weighted avg       1.00      1.00      1.00     18789
```

In [ ]: