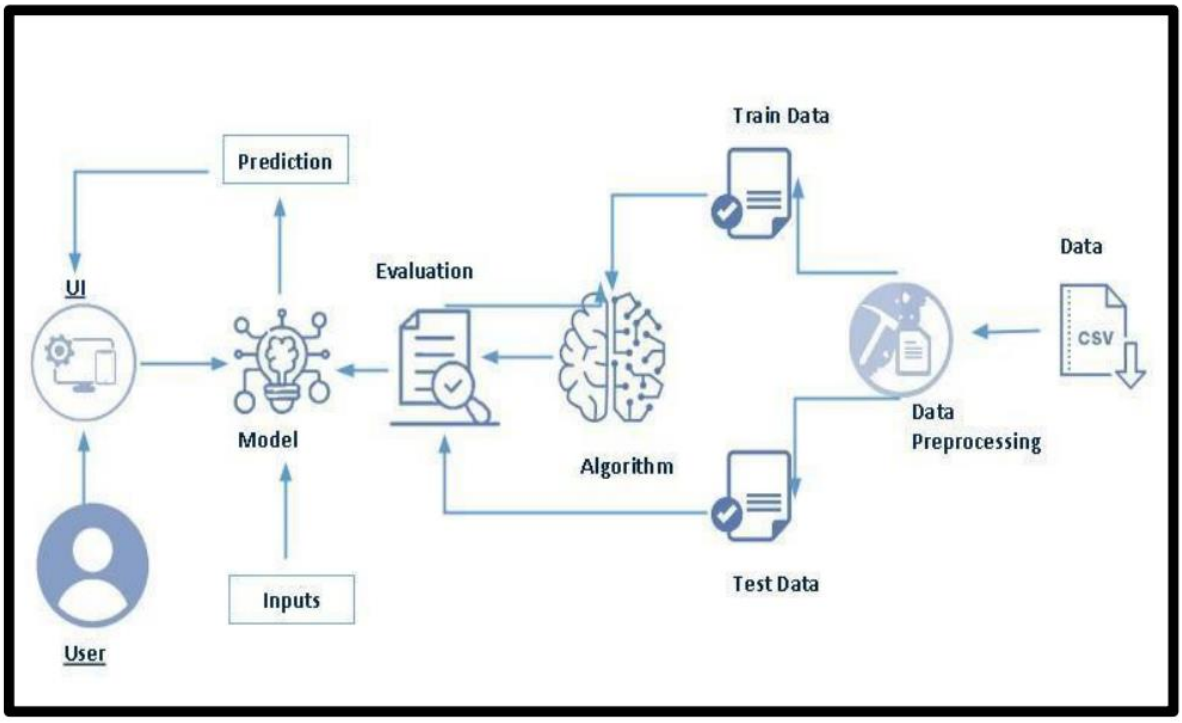
Acoustic Fire Extinguishing Prediction

Fire is one of the biggest hazards posed to humans due to high industrialization. This hazard can be controlled with the help of Fire Brigades and on smaller levels can be controlled with the help of fire extinguishers. The fire extinguishers are made up of chemicals and leave behind foam or chemicals after putting out the fire. These chemicals are harmful to humans and can also damage the equipment used in office creating higher loss.

The acoustic fire extinguisher used sound waves to put out fire. The fire can be curbed with the help of sound waves at certain frequencies. But there are limitations as to how big fire can be taken down with the acoustic fire extinguisher. Experiments were conducted where the researchers conducted experiment to put of fire. There were many changes made in each experiment like the frequency of sound waves, the distance of fire extinguisher, type of fuel, etc. The model we have built based on data can predict whether fire can be taken down with the acoustic fire extinguisher or no.

The model is right 96 out of 100 times on an average. This model should be used for checking whether fire can be taken down based on various parameters according to the industry needs. This model can tell whether acoustic fire extinguishers should be installed or no based on the type of environment.

Technical Architecture:



Project Flow:

- User is shown the Home page. The user will browse through Home page and click on the Predict button.
- After clicking the Predict button the user will be directed to the Details page where the user will input the parameters for the environment, they want to install acoustic fire extinguisher and click on the Predict button.
- User will be redirected to the Results page. The model will analyse the inputs given by the user and showcase the prediction whether acoustic fire extinguisher should be installed or no.

To accomplish this we have to complete all the activities listed below:

- Define problem / Problem understanding
    - Specify the business problem
    - Business Requirements
    - Literature Survey
    - Social or Business Impact
- Data Collection and Preparation
    - Collect the dataset
    - Data Preparation
    - Encoding data
    - Pre-processing of data
- Exploratory Data Analysis
    - Descriptive statistical
    - Visual Analysis
- Model Building
    - Creating a function for evaluation
    - Training and testing the Models using multiple algorithms
- Performance Testing & Hyperparameter Tuning
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning.
- Model Deployment
    - Save the best model
    - Integrate with Web Framework

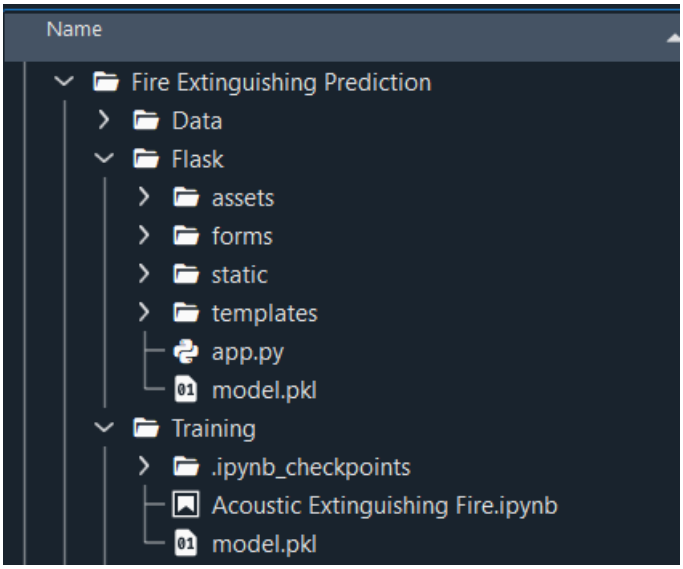
Prior Knowledge:

You must have the prior knowledge of the following topics to complete this project.

- ML Concepts:
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - K Nearest Neighbours: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - SVM: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classificationalgorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - Naïve Bayes: https://scikit-learn.org/stable/modules/naive_bayes.html
  - Logistic Regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
  - Gradient Boost: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metrics/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create project folder which contains files as shown below:



- The data obtained is in a excel file with extension .xlsx
- We are building a Flask application which will require the html files to be stored in the templates folder.

- The css files should be stored in the static folder.
- App.py file is used for routing purposes using scripting.
- Model.pkl is the saved model. This will further be used in the Flask integration.
- Training folder contains a model training file.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

Acoustic Fire Prediction involves predicting whether the fire can be extinguished with the help of acoustic fire extinguisher. The fire extinguishing process can be caused have various factors associated with them like the size of the cylinder, type of fuel, airflow, distance from extinguisher, etc. Acoustic Fire Extinguishing Prediction involves using machine learning algorithms to analyse large datasets and identify the underlying patterns in data. By analysing this data, machine learning algorithms can help in predicting whether the fire will be put out or no. If the fire can be put of with acoustic fire extinguisher, then installing it at offices or residential complexes is beneficial as there is no damage to the environment or people nearby due to the fire extinguishing process.

Activity 2: Business Requirements

Acoustic Fire extinguishing prediction project can have a variety of business requirements, depending on the specific goals and objectives of the project. Some potential requirements may include:

- Accurate and reliable information:
  The case of fire extinguishing prediction is critical and no false information can be tolerated since the consequences can be severe. Also there should not be any False Positives which means that our model predicts that the fire can be put out with the help of acoustic fire extinguisher but the fire does not go out with the help of acoustic fire extinguisher. This can cause large amounts of losses to the organization.
- Trust:
  Trust needs to be developed for the users to use the model. It is difficult to create trust among clients while dealing with a critical problem.
- Compliance:
  The model should be fit with all the relevant laws and regulations, such as OSHA Regulations for fire extinguishers.
- User friendly interface:
  The interface should be easy to use and understand by the user. The model should not ask inputs for which the user does not have answers.

Activity 3: Literature Survey

A literature survey for acoustic fire extinguishing project would involve researching and reviewing existing studies, articles, and other publications on the topic of disease prediction. The survey would aim to gather information on current

classification systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous fire extinguishing prediction projects, and any relevant data or findings that could inform the design and implementation of the current project.

Activity 4: Social or Business Impact.

Social Impact:

Whenever there is a fire, the people around and the environment does not get affected due to acoustic fire extinguisher. People can use this model to adapt to a safer and cleaner way to put out fire.

Business Impact:

Will positively impact the companies manufacturing the acoustic fire extinguishers. This will also help larger corporations and housing complexes to save money as the acoustic fire extinguisher does not need refill or does not expire.

Milestone 2: Data Collection and Preparation:

Machine Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .xlsx data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/code/jeyasrisenthil/flame-extinction-status-prediction/input

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the Libraries

Import the necessary libraries as shown in the image. Some of them are optional and can be skipped according to your usage.

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import pickle
        from sklearn.preprocessing import LabelEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score, confusion_matrix, precision_score

        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.svm import SVC
        from sklearn.naive_bayes import GaussianNB
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_excel() to read the dataset. As a parameter we have to give the directory of the csv file.

```
In [3]: data = pd.read_excel("C:/Users/hp/SB/Fire Extinguishing Prediction/Data/Acoustic_Extinguisher_Fire_Dataset.xlsx")

In [4]: data.head()
```

Out[4]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
0	1	gasoline	10	96	0.0	75	0
1	1	gasoline	10	96	0.0	72	1
2	1	gasoline	10	96	2.6	70	1
3	1	gasoline	10	96	3.2	68	1
4	1	gasoline	10	109	4.5	67	1

As we have two datasets, one for training and other for testing we will import both the csv files.

Activity 2: Data Preparation

As we have understood how the data is, let us pre-process the collected data.

The Machine Learning model cannot be trained on the imported data directly. The dataset might have randomness, we might have to clean the dataset and bring it in the right form. This activity involves the following steps:

- Removing Redundant Columns
- Handling Missing Values
- Encoding data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Removing Redundant Columns

There are no redundant columns, also the index column is right so there is no need to remove any column.

Activity 2.2: Handling Missing Values

```
In [58]: data.isnull().sum()

Out[58]: SIZE         0
         FUEL         0
         DISTANCE     0
         DESIBEL      0
         AIRFLOW      0
         FREQUENCY    0
         STATUS       0
         dtype: int64
```

There are no missing values in the dataset. That is why we can skip this step.

Activity 2.3: Encoding Data

The column Fuel has string data. The Machine Learning Models cannot be trained on string data, they need integer or float data. So we have to convert the string data to integer data, this is known as encoding. We will use Label encoder which will assign integer values to each unique string in the column.

```
In [31]: lb = LabelEncoder()
```

```
In [32]: data['FUEL'] = lb.fit_transform(data['FUEL'])
```

```
In [33]: data.head()
```

Out[33]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
0	1	0	10	96	0.0	75	0
1	1	0	10	96	0.0	72	1
2	1	0	10	96	2.6	70	1
3	1	0	10	96	3.2	68	1
4	1	0	10	109	4.5	67	1

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
In [7]: data.describe()
```

Out[7]:

	SIZE	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
count	17442.000000	17442.000000	17442.000000	17442.000000	17442.000000	17442.000000
mean	3.411765	100.000000	96.379142	6.975634	31.611111	0.497821
std	1.750977	54.773826	8.164096	4.736169	20.939149	0.500010
min	1.000000	10.000000	72.000000	0.000000	1.000000	0.000000
25%	2.000000	50.000000	90.000000	3.200000	14.000000	0.000000
50%	3.000000	100.000000	95.000000	5.800000	27.500000	0.000000
75%	5.000000	150.000000	104.000000	11.200000	47.000000	1.000000
max	7.000000	190.000000	113.000000	17.000000	75.000000	1.000000

Activity 2: Checking Unique values in each column

```
In [59]: data.value_counts()
```

```
Out[59]: SIZE  FUEL  DISTANCE  DESIBEL  AIRFLOW  FREQUENCY  STATUS
         1     0     10        75       0.8      1          1         1
         4     3     70        90       10.7     15         1         1
                             92       13.3     28         1         1
                                      11.2     19         1         1
                                      10.6     16         1         1
                                                                    ..
         2     3     130       102      4.4      48         0         1
                             103      3.7      51         0         1
                                      5.2      44         0         1
                                      5.6      40         0         1
         7     2     190       105      0.0      72         0         1
         Length: 17442, dtype: int64
```

Here we have seen the unique values for all the columns.

Activity 3: Visual Analysis

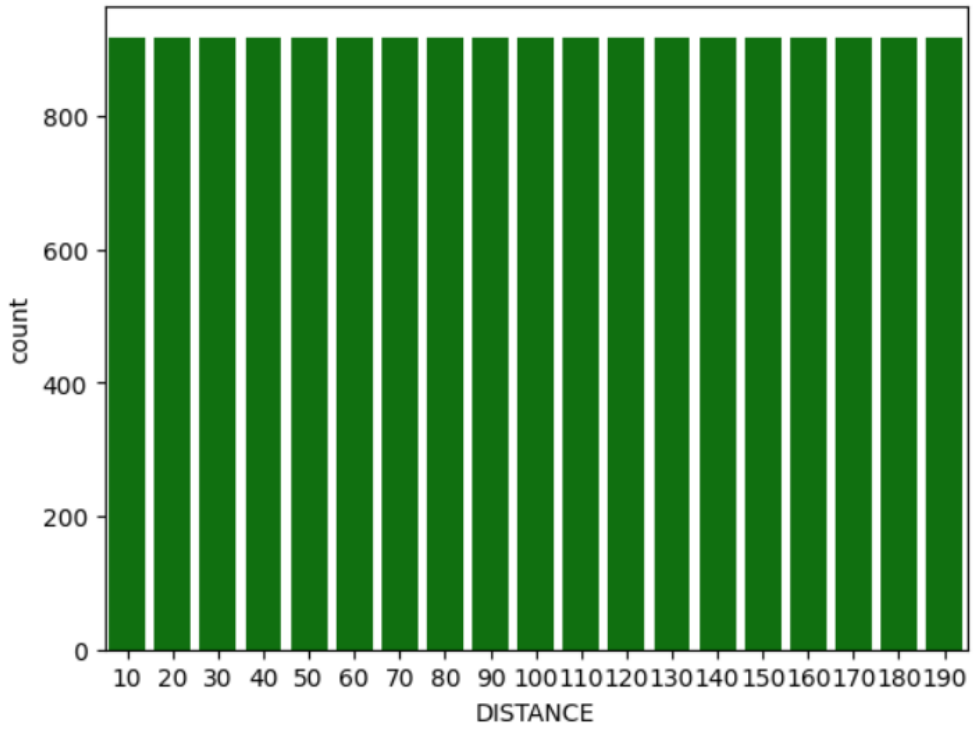
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 3.1: Univariate Analysis:

In simple words, univariate analysis is understanding the data with a single feature. We have displayed three different types of graphs and plots.

For simple visualizations we can use the seaborn library.

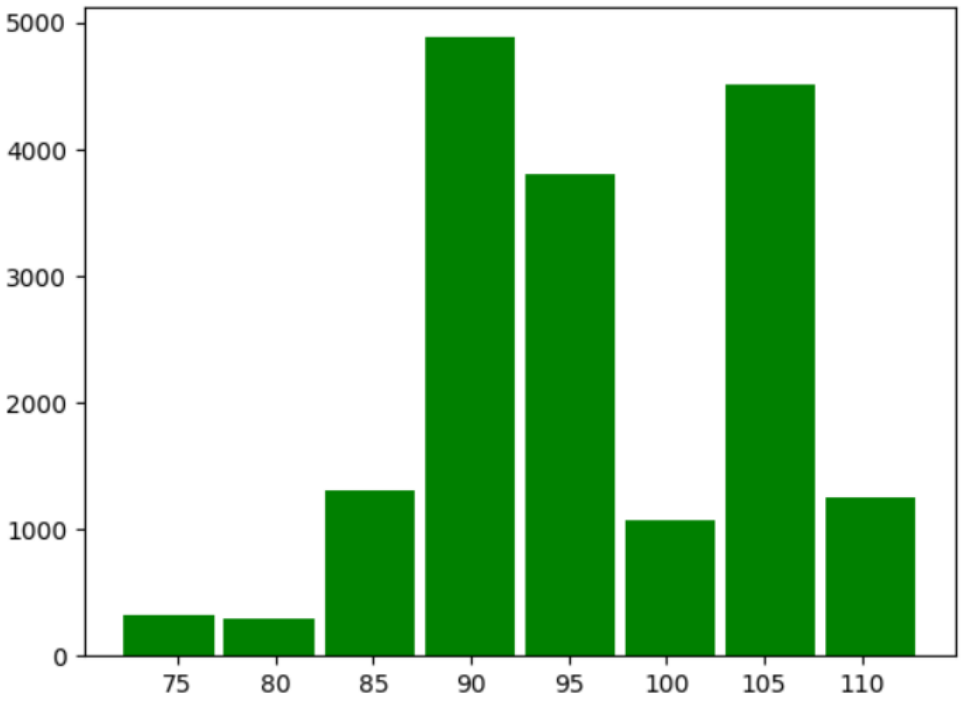
```
In [14]: sns.countplot(x='DISTANCE',data=data, color='green')
Out[14]: <Axes: xlabel='DISTANCE', ylabel='count'>
```



Here we can see that there are distances from 10 to 190 in the multiples of 10. We have plotted a countplot with the help of seaborn library. We have observed that there are equal number of observations for different distances.

```
In [18]: plt.hist(data['DESIBEL'], bins = 8, rwidth = 0.9,color = 'green')
```

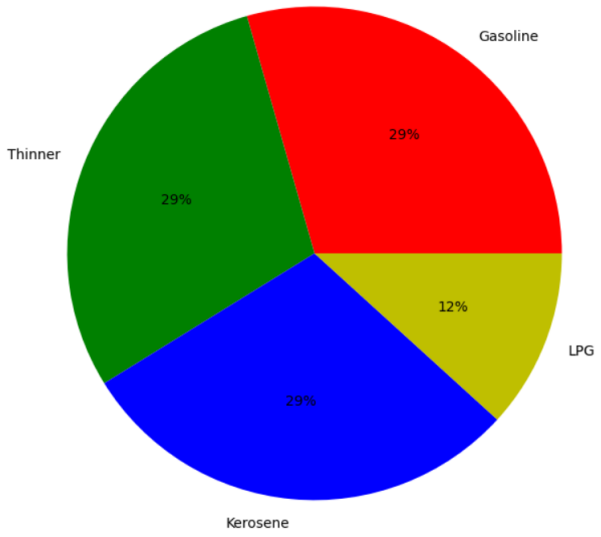
```
Out[18]: (array([ 323.,  289., 1309., 4879., 3808., 1071., 4505., 1258.]),
          array([ 72.   ,  77.125,  82.25 ,  87.375,  92.5  ,  97.625, 102.75 ,
                 107.875, 113.   ]),
          <BarContainer object of 8 artists>)
```



Here we have plotted histogram with the help of matplotlib library for the column DESIBEL. It shows the number of distributions for various decibel.

```
In [11]: plt.figure(figsize = (8,8))
         a = data['FUEL'].value_counts()
         plt.pie(x = a, data = data, labels= ['Gasoline','Thinner','Kerosene','LPG'], autopct='%.0f%%',colors = 'rgby')
         plt.title("Pie chart showing the distribution of Fuel of the Fire ")
```

```
Out[11]: Text(0.5, 1.0, 'Pie chart showing the distribution of Fuel of the Fire ')
```

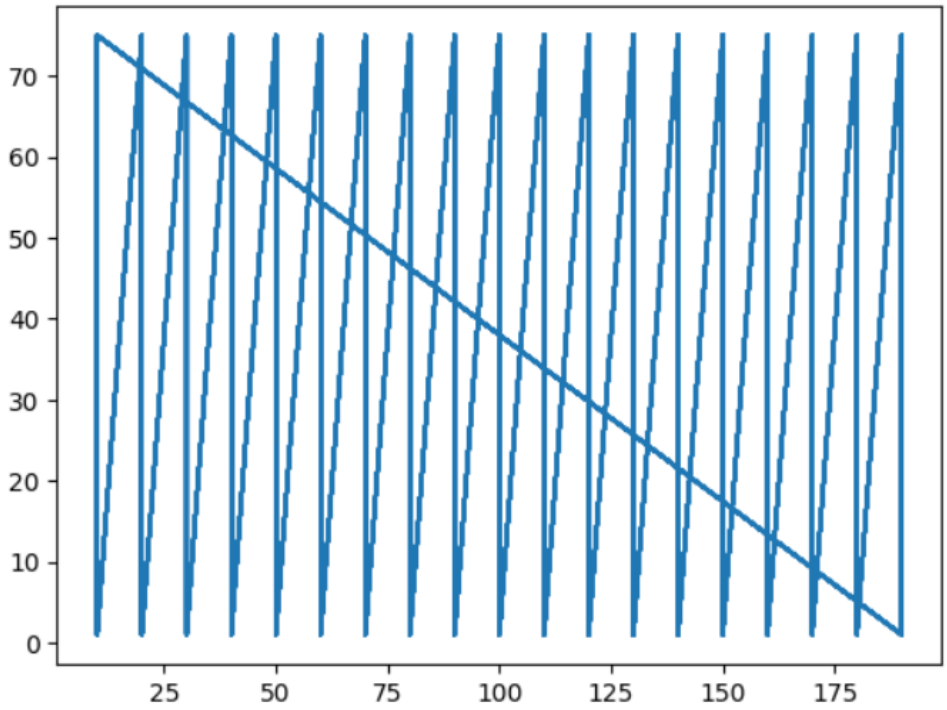


Here we have plotted a pie chart showing the distribution of different fuels in our data. We can see that there are equal number of observations for thinner, gasoline, Kerosene. For LPG fuel there are less number of observations.

Activity 3.2: Bivariate Analysis:

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between Distance feature and Frequency feature.

```
In [60]: plt.plot(data['DISTANCE'],data['FREQUENCY'])
Out[60]: [<matplotlib.lines.Line2D at 0x1633ee73eb0>]
```



We can see that as for each distance, the values of frequency increase gradually, until the next distance value is encountered. For each distance value the frequency varies from 10 Hz to 70 Hz.

Activity 3.3: Multivariate Analysis

In multivariate analysis we try to find the relation between multiple features. This can be done primarily with the help of Correlation matrix.

In [27]:
```
d1 = data[data['STATUS'] == 1]
d1.head()
```

Out[27]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
1	1	gasoline	10	96	0.0	72	1
2	1	gasoline	10	96	2.6	70	1
3	1	gasoline	10	96	3.2	68	1
4	1	gasoline	10	109	4.5	67	1
5	1	gasoline	10	109	7.8	66	1

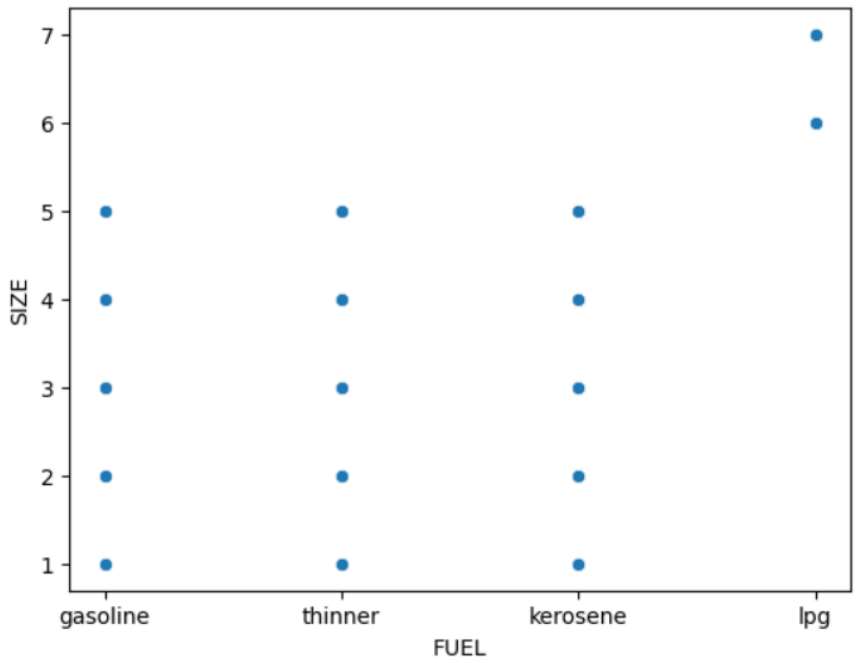
In [28]:
```
d2 = data[data['STATUS'] == 0]
d2.head()
```

Out[28]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
0	1	gasoline	10	96	0.0	75	0
54	1	gasoline	20	96	0.0	75	0
55	1	gasoline	20	96	0.0	72	0
107	1	gasoline	20	76	1.0	1	0
108	1	gasoline	30	106	0.0	75	0

Here we have created two datasets, one for The extinction state and one for The non-extinction state of the fire using Boolean indexing.

```
In [30]: sns.scatterplot(data= d1, x='FUEL', y = 'SIZE')
Out[30]: <Axes: xlabel='FUEL', ylabel='SIZE'>
```



We create visualizations for the fire extinction state. We create scatterplot with the help of the seaborn library for the columns Fuel vs Size.

We are looking for any clear differentiation for the fire to be in The extinction or non-extinction state.

Activity 4: Pre-processing of data

This is where we have to split the data and make it suitable for training machine learning models.

Activity 4.1: Split data into Features and Target Variable

We have checked the data and are sure to train machine learning models with this data.

We first need to separate the features and the target variable. The features are used to predict the target variable.

```
In [35]: X = data.drop('STATUS',axis = 1)
         y = data['STATUS']
```

```
In [61]: print(X.shape)
         print(y.shape)

         (17442, 6)
         (17442,)
```

Activity 4.2: Split data into training and testing data

Now we need to split the training data into training and validation data. It can be done using the following command.

```
In [36]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

We have kept 80 % data for training and 20% is used for testing.

Milestone 4: Model Building

Activity 1: Creating a function for model evaluation

We will be creating multiple models and then testing them. It will reduce our monotonous task if we directly write a function for model evaluation.

```
In [37]: def model_evaluation(classifier):
             cm = confusion_matrix(y_test, classifier.predict(X_test))
             counts = [value for value in cm.flatten()]
             labels = [f'{v1}' for v1 in counts]
             labels = np.asarray(labels).reshape(2,2)
             sns.heatmap(cm, annot = labels, cmap = 'Greens', fmt = '')
             y_pred = classifier.predict(X_test)
             yt_pred = classifier.predict(X_train)
             print('The Training Accuracy of the algorithm is ', accuracy_score(y_train, yt_pred))
             print('The Testing Accuracy of the algorithm is ', accuracy_score(y_test, y_pred))
             return [(accuracy_score(y_train * 100, yt_pred * 100) * 100), (accuracy_score(y_test * 100, y_pred * 100) * 100), precision_s
```

This function will show the accuracies of prediction of model for training and testing data. It will also the return those accuracies along with the precision score because precision score is used to check for False Positives.

Activity 2: Training and testing the models using multiple algorithms

Now that we have clean data, a function for evaluation it is time to build models to train the data. For this project we will be using 7 different binary classification algorithms to build our models. The best model will be used for prediction.

Activity 2.1: K Nearest Neighbors Model

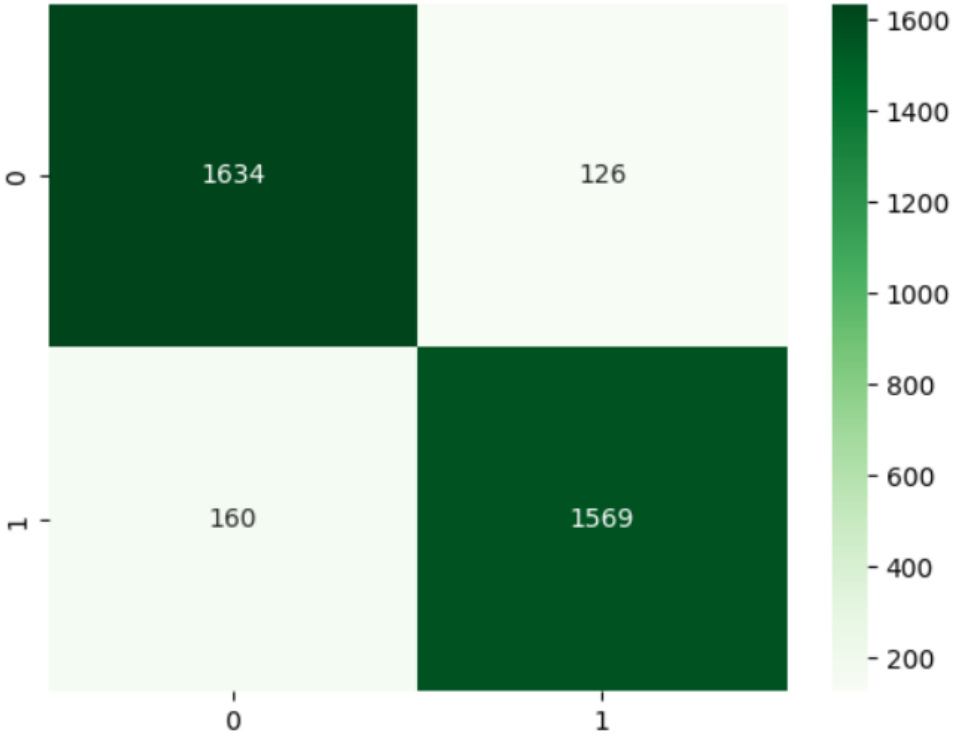
A variable is created with name knn which has KNeighborsClassifier() algorithm initialised in it. The knn model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [38]: knn = KNeighborsClassifier()
         knn.fit(X_train, y_train)
```

```
Out[38]: KNeighborsClassifier
         KNeighborsClassifier()
```

```
In [39]: knn_r = model_evaluation(knn)
```

```
The Training Accuracy of the algorithm is  0.9434530208557299
The Testing Accuracy of the algorithm is  0.9180280882774434
```



Here we can see that the model has achieved 94% accuracy on training data, and 91% accuracy on testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named knn_r.

Activity 2.2: SVM Model

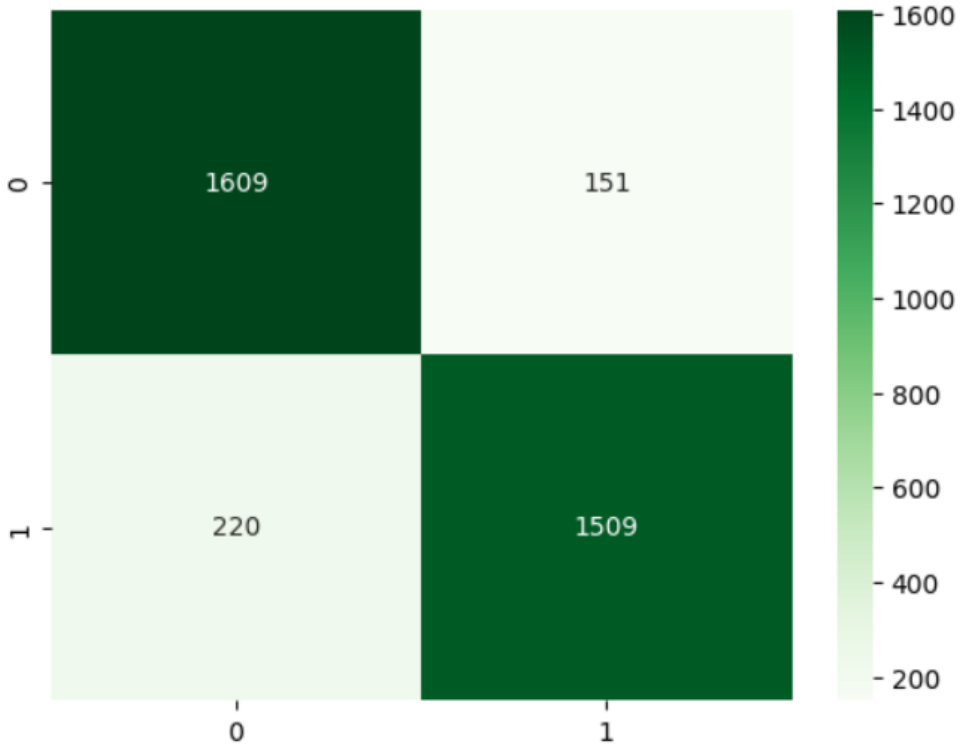
A variable is created with name Svm which has SVC() algorithm initialised in it. The svm model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [40]: Svm = SVC()
         Svm.fit(X_train, y_train)
```

```
Out[40]: SVC()
```

```
In [41]: Svm_r = model_evaluation(Svm)
```

```
The Training Accuracy of the algorithm is  0.8893427936644449
The Testing Accuracy of the algorithm is  0.8936658068214388
```



Here we can see that the model has achieved 88% accuracy on training data and 89% accuracy on testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named svm_r.

Activity 2.3: Naïve Bayes

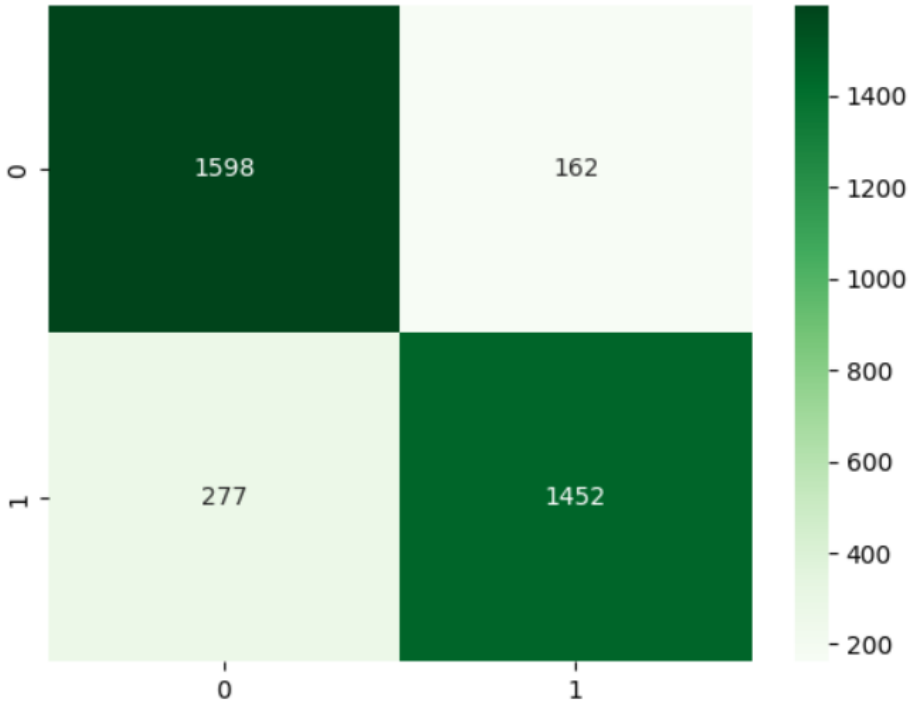
A variable is created with name gnb which has GaussianNB() algorithm initialised in it. The gnb model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [42]: gnb = GaussianNB()
         gnb.fit(X_train, y_train)
```

```
Out[42]: ▾ GaussianNB
         GaussianNB()
```

```
In [43]: gnb_r = model_evaluation(gnb)
```

```
The Training Accuracy of the algorithm is  0.8705654697914427
The Testing Accuracy of the algorithm is  0.8741759816566351
```



Here we can see that the model has achieved 87% accuracy on training data and 87% accuracy on testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named gnb_r.

Activity 2.4: Logistic Regression

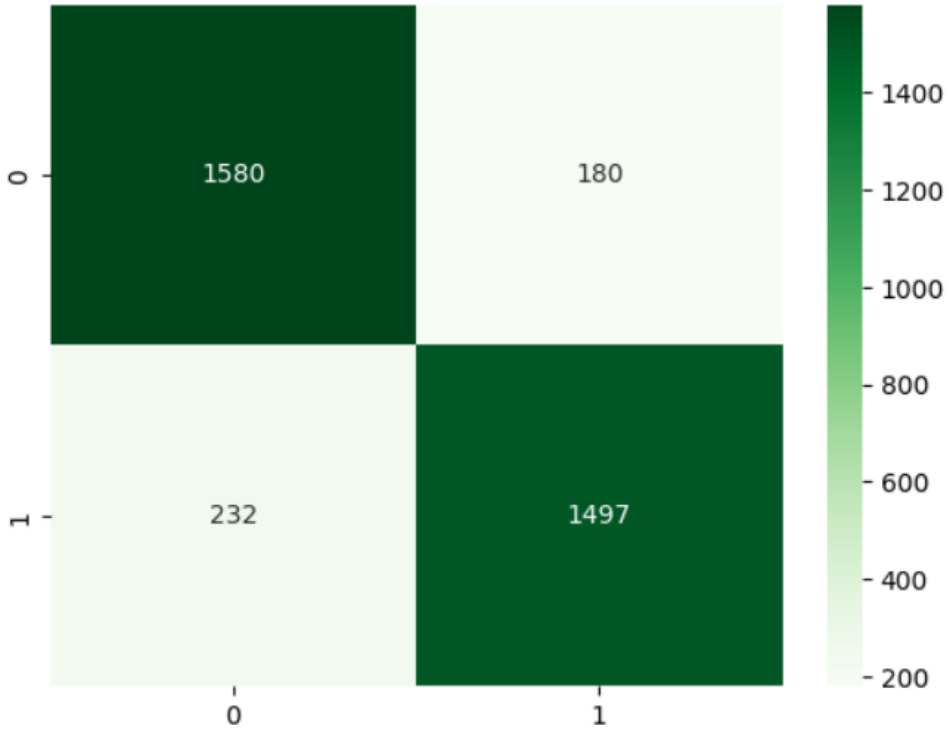
A variable is created with name lr which has LogisticRegression() algorithm initialised in it. The lr model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [44]: lr = LogisticRegression()
         lr.fit(X_train, y_train)
```

```
Out[44]: LogisticRegression
         LogisticRegression()
```

```
In [45]: lr_r = model_evaluation(lr)
```

```
The Training Accuracy of the algorithm is  0.874292266896008
The Testing Accuracy of the algorithm is  0.881914588707366
```



Here we can see that the model has achieved 87% accuracy on training data and 88% accuracy on testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named lr_r.

Activity 2.5: Decision Tree Model

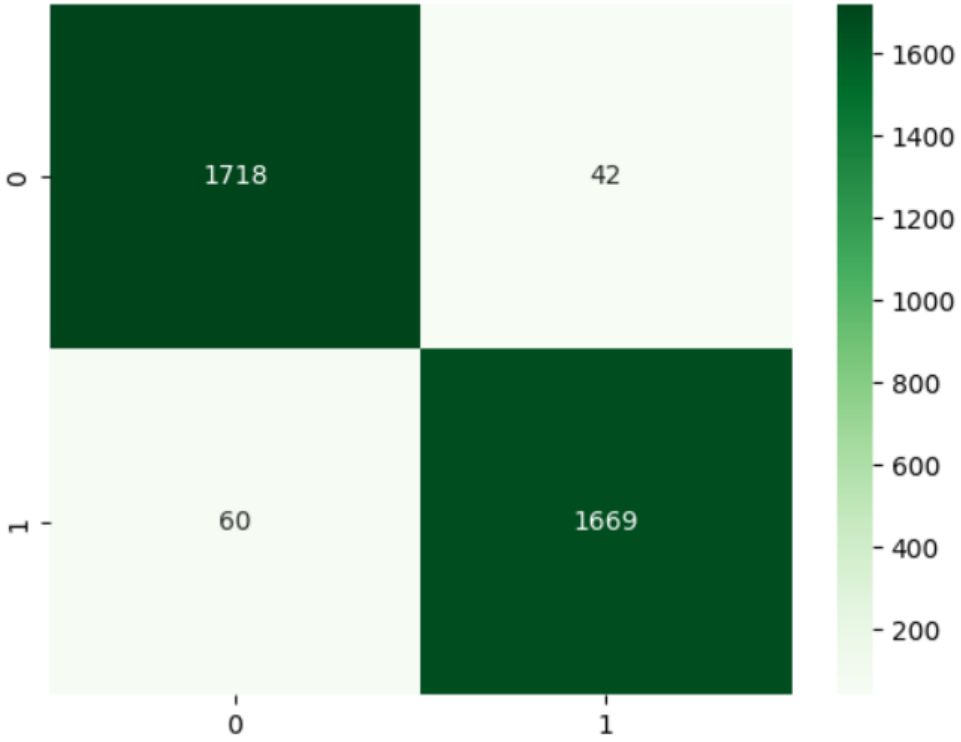
A variable is created with name dt which has DecisionTreeClassifier() algorithm initialised in it with a parameter max_depth set to 11. The dt model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [46]: dt = DecisionTreeClassifier(max_depth= 11)
         dt.fit(X_train, y_train)
```

```
Out[46]: DecisionTreeClassifier
         DecisionTreeClassifier(max_depth=11)
```

```
In [47]: dt_r = model_evaluation(dt)
```

```
The Training Accuracy of the algorithm is  0.98602451085788
The Testing Accuracy of the algorithm is  0.9707652622527945
```



Here we can see that the model has achieved 98% accuracy on training and has achieved 97% accuracy for testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named dt_r.

Activity 2.6: Random Forest Model

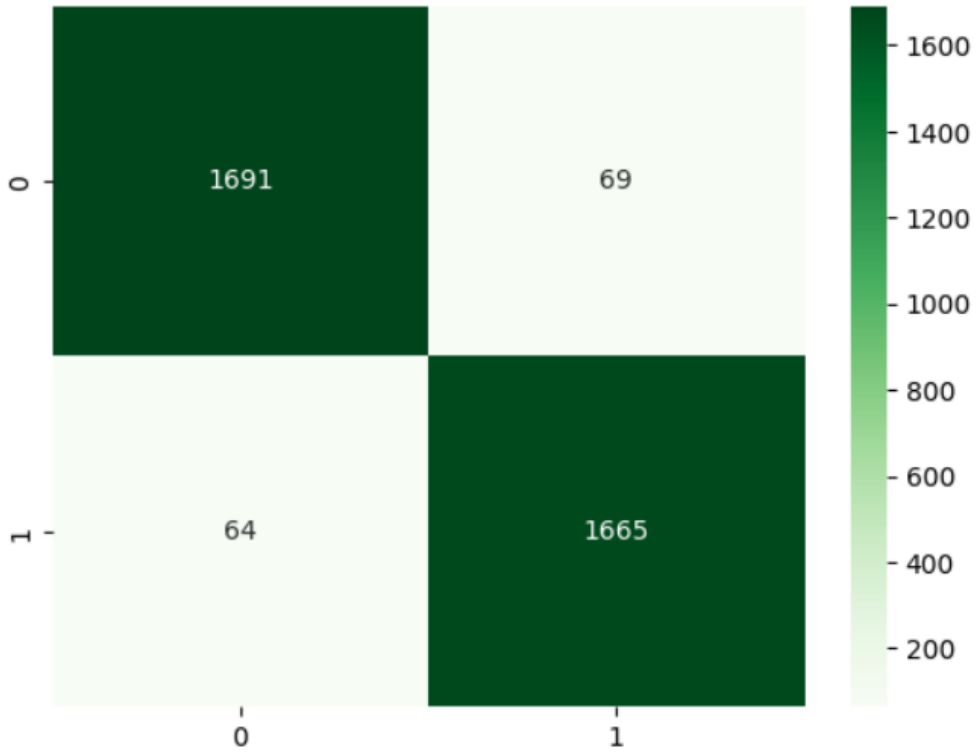
Random Forest Classifier is a Bagging model which utilises multiple decision trees and takes their aggregate to give a prediction. A variable is created with name rf which has RandomForestClassifier() algorithm initialised in it with a parameter max_depth set to 11. The rf model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [48]: rf = RandomForestClassifier(max_depth=11)
         rf.fit(X_train, y_train)
```

```
Out[48]: RandomForestClassifier
         RandomForestClassifier(max_depth=11)
```

```
In [49]: rf_r = model_evaluation(rf)
```

```
The Training Accuracy of the algorithm is  0.9873862251845481
The Testing Accuracy of the algorithm is  0.9618801948982516
```



Here we can see that the model has achieved 98.7% accuracies on training data and has achieved 96.1% accuracy for testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named rf_r.

Activity 2.7: Gradient Boosting Model

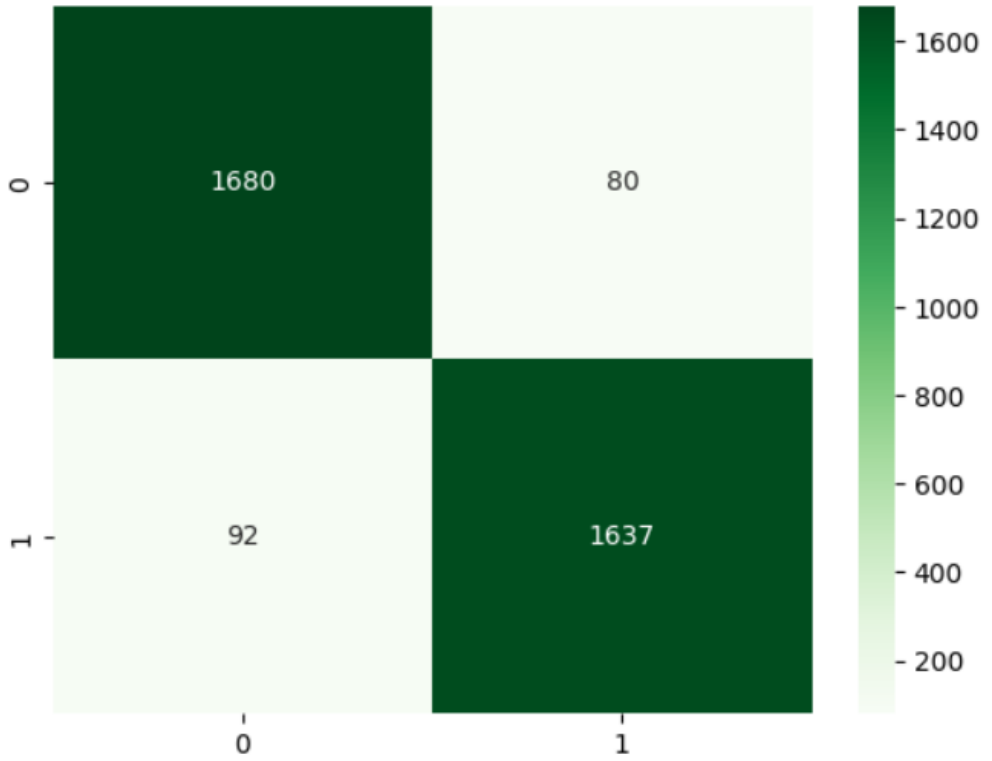
Random Forest Classifier is a Bagging model which utilises multiple decision trees and takes their aggregate to give a prediction. A variable is created with name rf which has RandomForestClassifier() algorithm initialised in it with a parameter max_depth set to 11. The rf model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [50]: gb = GradientBoostingClassifier()
         gb.fit(X_train, y_train)
```

```
Out[50]: GradientBoostingClassifier
         GradientBoostingClassifier()
```

```
In [51]: gb_r = model_evaluation(gb)
```

```
The Training Accuracy of the algorithm is  0.9530566903174944
The Testing Accuracy of the algorithm is  0.9507022069360849
```



Here we can see that the model has achieved 95.3% accuracies on training data and has achieved 95% accuracy for testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named gb_r.

Milestone 5 : Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with Multiple Evaluation metrics

The problem statement given to us cannot have False Positives. This will create a loss of huge amount of resources. We can use Precision Score for the same purpose. We have training and testing accuracies along with Precision Score as the Evaluation Metrics.

In [52]:
```
results = pd.DataFrame(data = [knn_r, Svm_r, gnb_r, lr_r,dt_r, rf_r, gb_r],
                       columns = ['Training Accuracy', 'Testing Accuracy', 'Precision Score'],
                        index = ['K Nearest Neighbors', 'Support Vector Machines',
                                 'Naive Bayes', 'Logistic Regression',
                                 'Decision Tree', 'Random Forest', 'Gradient Boost'])
```

In [53]:
```
results
```

Out[53]:

	Training Accuracy	Testing Accuracy	Precision Score
K Nearest Neighbors	94.345302	91.802809	0.925664
Support Vector Machines	88.934279	89.366581	0.909036
Naive Bayes	87.056547	87.417598	0.899628
Logistic Regression	87.429227	88.191459	0.892665
Decision Tree	98.602451	97.076526	0.975453
Random Forest	98.738623	96.188019	0.960208
Gradient Boost	95.305669	95.070221	0.953407

From the table we can see that Decision Tree and Random Forest models perform the best in terms of accuracies and also in terms of the Precision Score.

Activity 2: Comparing model accuracy before and after applying hyperparameter tuning

As the accuracies are already so high, we need not do hyperparameter tuning for the models.

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and to be able to use it in the future**.**

After checking the performance, we decide to save the rf model.

```
In [57]: pickle.dump(rf, open('model.pkl','wb'))
```

We save the model using the pickle library into a file named model.pkl

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

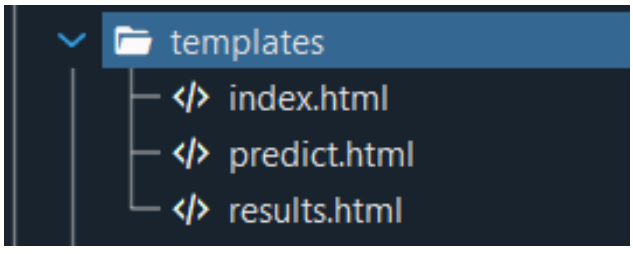
- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building HTML pages:

For this project we create three HTML files namely

- Index.html
- Details.html
- Results.html

And we will save them in the templates folder.



Activity 2.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

Import the necessary Libraries.

```
from flask import Flask, render_template, request
import pandas as pd
import pickle
```

This code first loads the saved Random Forest model from the "model.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

```
model = pickle.load(open('model.pkl','rb'))
app = Flask(__name__)
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.

The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

The "render_template()" method is used to render an HTML template named "index.html". The “index.html” is the home page.

```
@app.route("/")
def home():
    return render_template('index.html')
```

The route in this case is "/details". When a user accesses the "/predict" route of the website, this function is “index()” called. The "render_template()" method is used to render an HTML template named "details.html".

```
@app.route('/details')
def pred():
    return render_template('details.html')
```

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/predict", and the method is set to GET and POST.

The function "predict()" is then associated with this route. In the details.html page we are going to take inputs from the user, which will be in the form of dropdowns and input text for the integer values.

The values are stored in request.form.values()

We take out each value with the associated label for it.

Eg: request.form['SIZE'] will give us the value entered by user for the size dropdown

This prediction is returned to the results.html page using render_template()

```
@app.route('/predict',methods = ['GET','POST'])
def predict():
    SIZE = request.form['SIZE']
    FUEL = request.form['FUEL']
    DISTANCE = request.form['DISTANCE']
    DESIBEL = request.form['DESIBEL']
    AIRFLOW = request.form['AIRFLOW']
    FREQUENCY = request.form['FREQUENCY']

    total = [[SIZE,FUEL,DISTANCE,DESIBEL,AIRFLOW,FREQUENCY]]
    d1 = pd.DataFrame(data = total, columns = ['SIZE','FUEL','DISTANCE','DESIBEL','AIRFLOW','FREQUENCY'])
    prediction = model.predict(d1)
    prediction = prediction[0]
    if prediction == 0:
        return render_template('results.html', prediction_text = "The fire is in non extinction state")
    else:
        return render_template('results.html', prediction_text = "The fire is in extinction state.")
```

Main Function:

This code sets the entry point of the Flask application. The function “app.run()” is called, which starts the Flask deployment server.

```
if __name__ == "__main__":
    app.run()
```

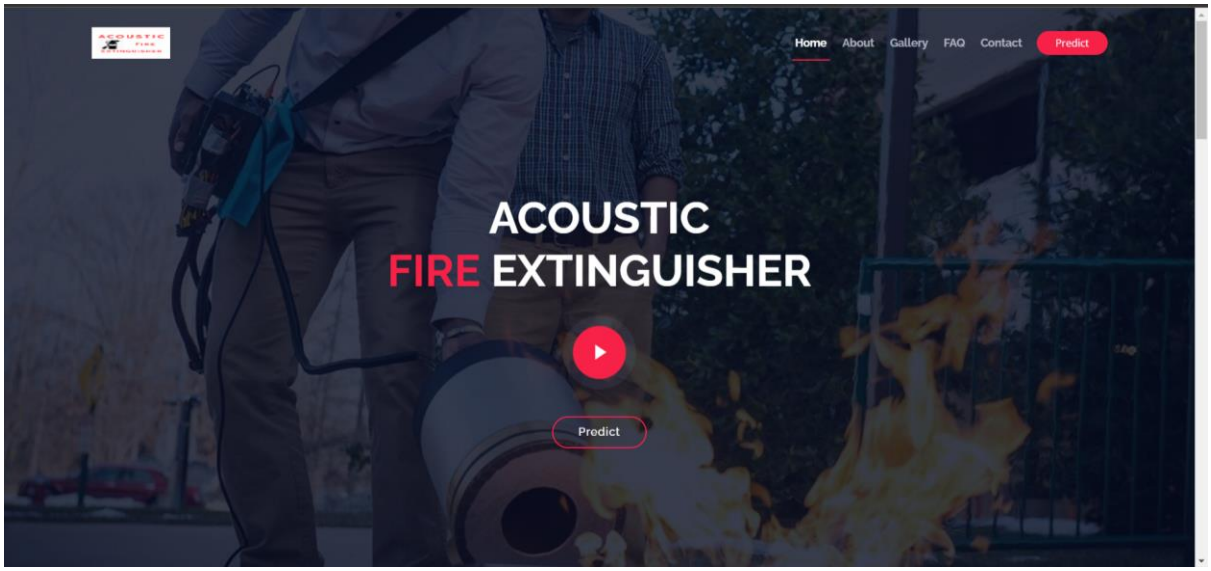
Activity 2.3: Run the Web Application

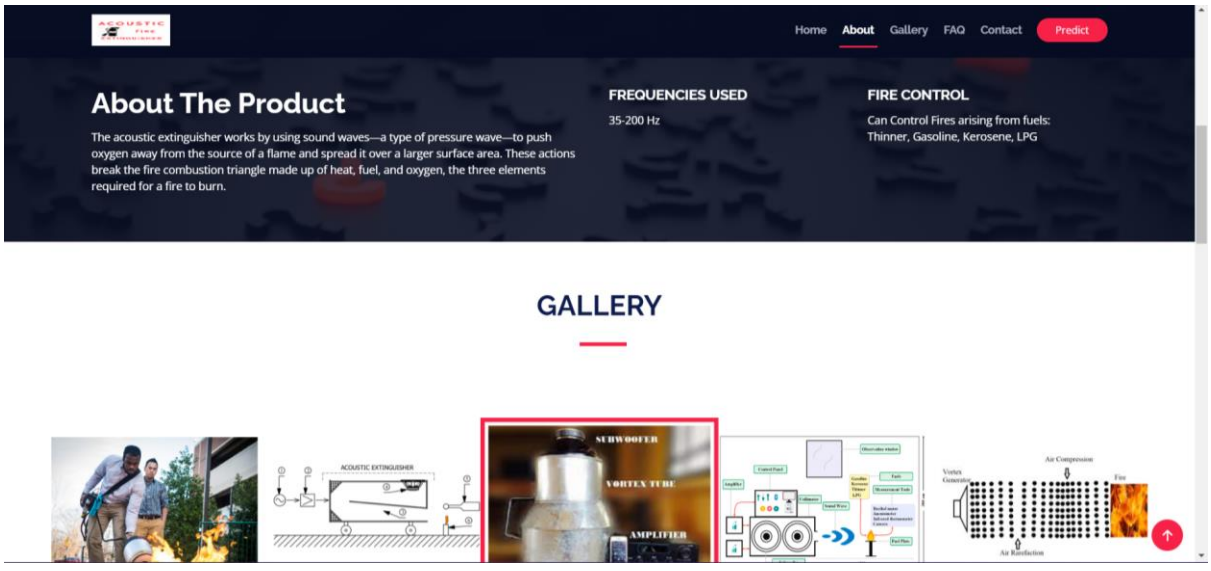
When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser.

```
In [1]: runfile('C:/Users/hp/SB/Fire Extinguishing Prediction/Flask/
app.py', wdir='C:/Users/hp/SB/Fire Extinguishing Prediction/Flask')
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
```

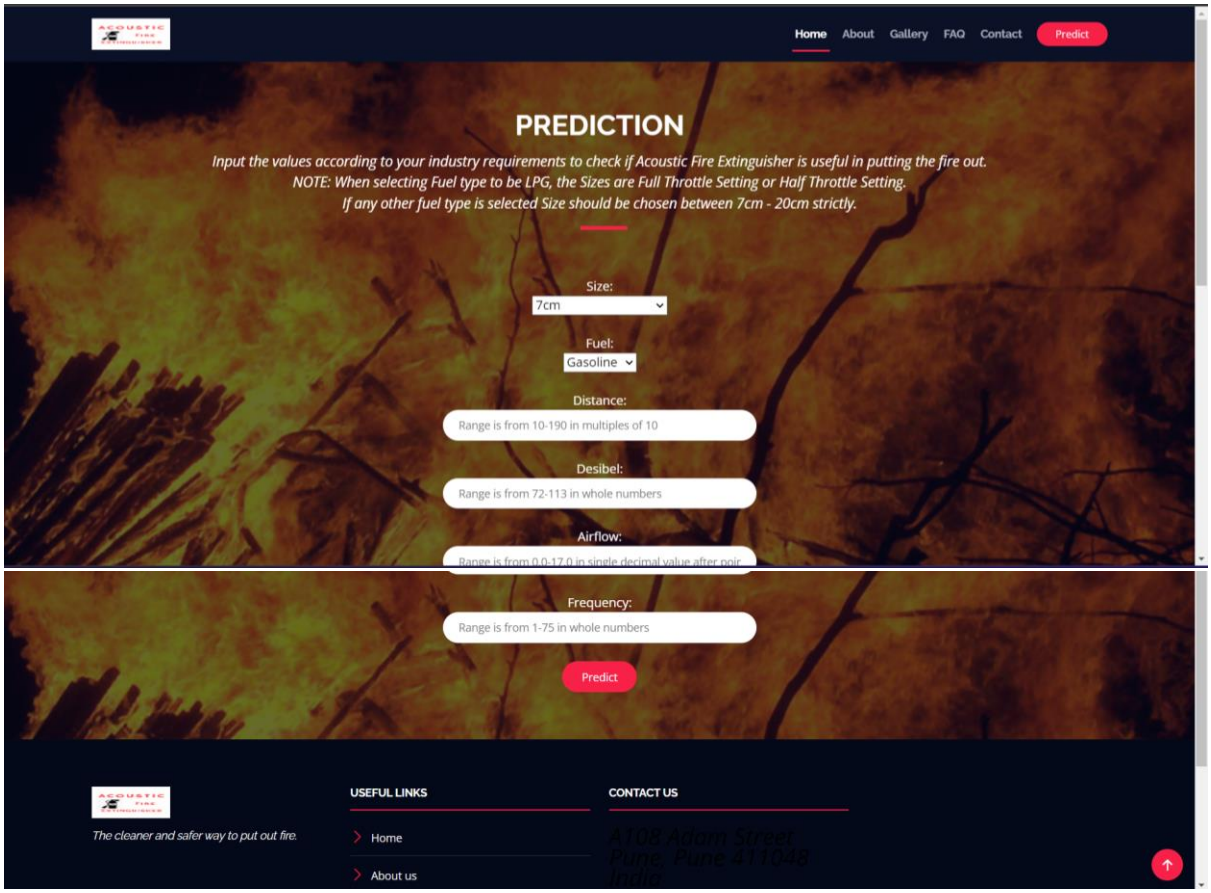
When we paste the URL in a web browser, our index.html page will open. It contains various sections in the header bar such as Home, Predict, About Model, Testimonials, FAQ, Contact. There is some information given on the web page about our model.

If you click on the Predict button on home page or in the header bar you will be redirected to the details.html page.





Our Detials.html looks as shown below.



We are provided with 6 input fields out of which 2 are dropdowns and the rest 4 are input fields where user will input integer values with the instructions provided .

We will input values as follows:

Size: 14cm

Fuel: Kerosene

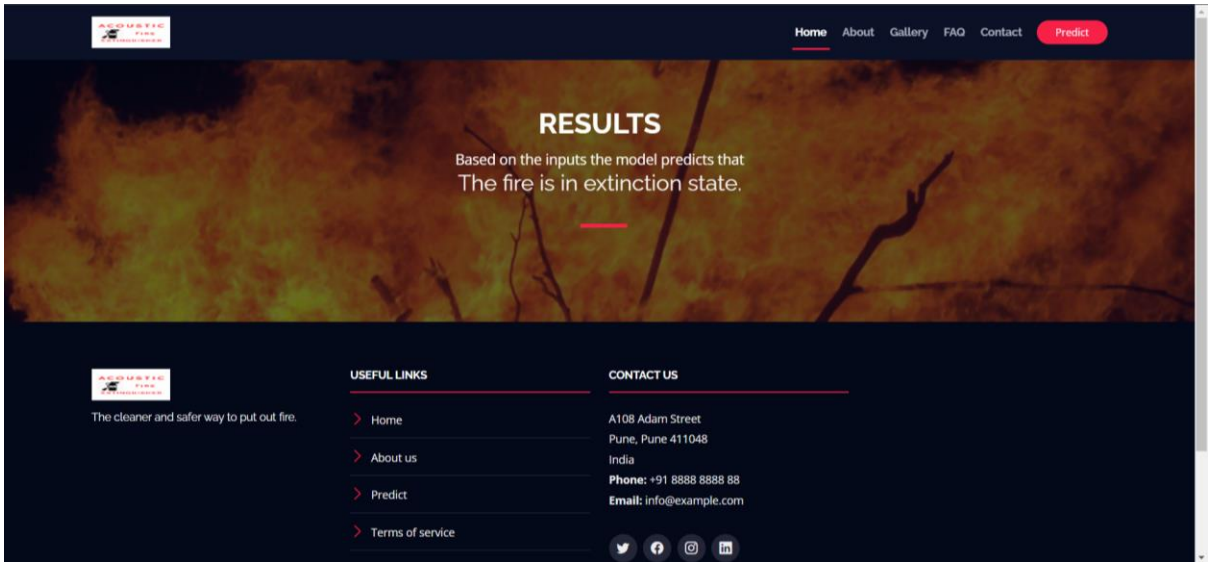
Distance: 40

Desibel: 80

Airflow: 5.0

Frequency: 45

We will be redirected to the Results.html page once we click the Predict button.



The results say that the fire is in the extinction state based on the various inputs we have given.