Mine Explosion Prediction

Coal mine explosions are rare, but they cause the most fatalities. Nearly 8,000 lives have been lost in US coal mines alone. Worldwide figures are much higher.

There are two main types of coal mine explosions: methane explosions and coal dust explosions.

Methane explosions occur when a buildup of methane gas contacts a heat source and there is not enough air to dilute the gas level below its explosion point.

Methane is formed as a byproduct of the coal formation. The methane that is adsorbed in the coal is released as the coal is mined or it migrates from surrounding sources above or below the coal seam through fractures created by the coal extraction process.
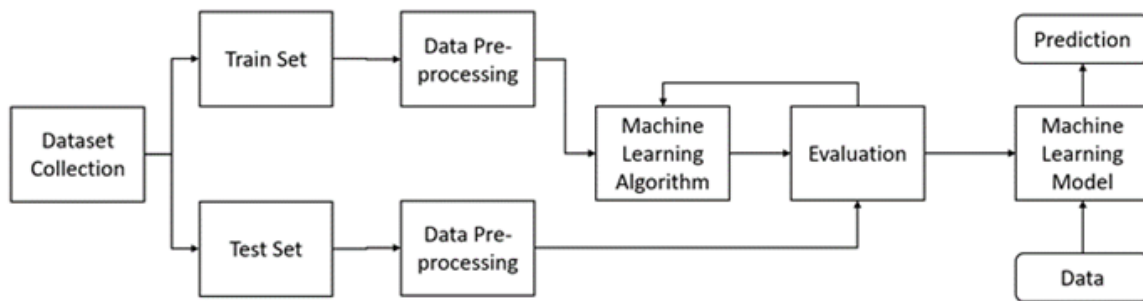
Coal dust is a fine powdered form of coal, which is created by the crushing, grinding, or pulverizing of coal. Because of the brittle nature of coal, coal dust can be created during mining, transportation, or by mechanically handling coal. It is a form of fugitive dust.

Grinding coal to dust before combusting it improves the speed and efficiency of burning and makes the coal easier to handle. However, coal dust is hazardous to workers if it is suspended in air outside the controlled environment of grinding and combustion equipment. It poses the acute hazard of forming an explosive mixture in air and the chronic hazard of causing pulmonary illness in people who inhale excessive quantities of it.

Block Diagram:

```
┌──────────┐      ┌──────────┐      ┌──────────────┐      ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐
│ Emission │      │ Machine  │      │ Prediction on│      │ Triggers alarm when limit │
│   Data   │ ───▶ │ Learning │ ───▶ │  screen for  │ ───▶ │  exceeds normal level     │
│          │      │  Model   │      │  monitoring  │      └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
└──────────┘      └──────────┘      └──────────────┘
```

Machine Learning Workflow:

```
                    ┌──────────┐   ┌──────────────┐                                         ┌────────────┐
                    │ Train Set│──▶│   Data Pre-  │                                         │ Prediction │
                ┌──▶│          │   │  processing  │──┐                                      └────────────┘
                │   └──────────┘   └──────────────┘  │                                            ▲
┌────────────┐  │                                    │   ┌──────────┐    ┌──────────┐    ┌────────────┐
│  Dataset   │──┤                                    └──▶│ Machine  │───▶│          │───▶│  Machine   │
│ Collection │  │                                        │ Learning │    │Evaluation│    │ Learning   │
└────────────┘  │                                    ┌──▶│Algorithm │    │          │    │   Model    │
                │   ┌──────────┐   ┌──────────────┐  │   └──────────┘    └──────────┘    └────────────┘
                └──▶│ Test Set │──▶│   Data Pre-  │──┘                                          ▲
                    │          │   │  processing  │                                       ┌────────────┐
                    └──────────┘   └──────────────┘                                       │    Data    │
                                                                                          └────────────┘
```

Project Flow:

- User interacts with the UI (User Interface) to enter the current mine working conditions.

- Entered readings are analyzed and predictions are made based on interpretation that whether mine will explode or situation will lead to explosion.

- Predictions are popped onto the UI.

Prerequisites:

1. **To develop this project, we need to install following software/packages:**

Anaconda Navigator:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder code.

2. **To build a Mine explosion prediction**

● **Install sci-kit learn**

Scikit-learn (**Sklearn**) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistence interface in Python.

● **Install Flask**

○ Flask is a Web application framework written in python

○ For installation of Flask

■ Open your anaconda prompt and Type "pip install Flask"

To accomplish this, we must complete all the activities and tasks listed below

1. Data Collection

Download dataset /create dataset:

For this project the dataset has been downloaded from kaggle.com.

2. Data Pre-processing

Importing required Libraries:

importing packages

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.preprocessing import LabelEncoder
        from sklearn.preprocessing import OneHotEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import r2_score
        from sklearn.metrics import accuracy_score
        from sklearn.svm import SVC
        from sklearn.metrics import confusion_matrix
```

Pandas: It is a python library mainly used for data manipulation.
NumPy: This python library is used for numerical analysis.
Matplotlib and Seaborn: Both are the data visualization library used for plotting graph which will help us for understanding the data.

Accuracy score: used in classification type problem and for finding accuracy it is used.

R2 Score: Coefficient of Determination or R^2 is another metric used for evaluating the performance of a regression model. The metric helps us to compare our current model with a constant baseline and tells us how much our model is better.

Train_test_split: used for splitting data arrays into training data and for testing data.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistence interface in Python.

Importing the dataset:

importing data set

```
In [2]: data=pd.read_csv(r"C:\Users\sunkari's\Desktop\exa\seismic-bumps.csv")
```

```
In [3]: data.head()
```

Out[3]:

	seismic	seismoacoustic	shift	genergy	gpuls	gdenergy	gdpuls	ghazard	nbumps	nbumps2	nbumps3	nbumps4	nbumps5	nbumps6	nbumps7	nbum
0	a		a	N	15180	48	-72	-72	a	0	0	0	0	0	0	0
1	a		a	N	14720	33	-70	-79	a	1	0	1	0	0	0	0
2	a		a	N	8050	30	-81	-78	a	0	0	0	0	0	0	0
3	a		a	N	28820	171	-23	40	a	1	0	1	0	0	0	0
4	a		a	N	12640	57	-63	-52	a	0	0	0	0	0	0	0

- You might have your data in .csv files, .excel files or .tsv files or something else. But the goal is the same in all cases. If you want to analyse that data using pandas, the first step will be to read it into a data structure that's compatible with pandas.

- Let's load a .csv data file into pandas. There is a function for it, called **read_csv().**We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).

Data Visualization:

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods and used for determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

- To check first five rows of dataset, we have a function call **head( ).**

```
In [42]: data.head(10)
Out[42]:
```

	seismic	seismoacoustic	shift	genergy	gpuls	gdenergy	gdpuls	ghazard	nbumps	nbumps2	nbumps3	nbumps4	nbumps5	nbumps6	nbumps7	nbum
0	0	0	0	15180	48	-72	-72	0	0	0	0	0	0	0	0
1	0	0	0	14720	33	-70	-79	0	1	0	1	0	0	0	0
2	0	0	0	8050	30	-81	-78	0	0	0	0	0	0	0	0
3	0	0	0	28820	171	-23	40	0	1	0	1	0	0	0	0
4	0	0	0	12640	57	-63	-52	0	0	0	0	0	0	0	0
5	0	0	1	63760	195	-73	-65	0	0	0	0	0	0	0	0
6	0	0	1	207930	614	-6	18	0	2	2	0	0	0	0	0
7	0	0	0	48990	194	-27	-3	0	1	0	1	0	0	0	0
8	0	0	0	100190	303	54	52	0	0	0	0	0	0	0	0
9	0	0	1	247620	675	4	25	0	1	1	0	0	0	0	0

- This head () function returns the first 5 rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

- To check last five rows of dataset, we have a function call **tail().**

In [43]: data.tail(10)

Out[43]:

	seismic	seismoacoustic	shift	genergy	gpuls	gdenergy	gdpuls	ghazard	nbumps	nbumps2	nbumps3	nbumps4	nbumps5	nbumps6	nbumps7	nk
2568	1	0	1	27740	443	81	41	0	0	0	0	0	0	0	0
2569	1	0	1	15460	249	15	-2	0	0	0	0	0	0	0	0
2570	1	0	1	22060	275	64	8	0	0	0	0	0	0	0	0
2571	1	0	1	16330	270	7	-14	0	0	0	0	0	0	0	0
2572	1	0	1	28910	307	115	20	0	0	0	0	0	0	0	0
2573	1	0	1	81410	785	432	151	1	0	0	0	0	0	0	0
2574	1	0	1	42110	555	213	118	0	0	0	0	0	0	0	0
2575	1	0	1	26960	540	101	112	0	0	0	0	0	0	0	0
2576	0	0	1	16130	322	2	2	0	0	0	0	0	0	0	0
2577	0	0	1	12750	235	-10	-10	0	0	0	0	0	0	0	0

- For finding the names of the columns present in the dataset we make use of **columns**

In [6]: data.columns

Out[6]: Index(['seismic', 'seismoacoustic', 'shift', 'genergy', 'gpuls', 'gdenergy',
       'gdpuls', 'ghazard', 'nbumps', 'nbumps2', 'nbumps3', 'nbumps4',
       'nbumps5', 'nbumps6', 'nbumps7', 'nbumps89', 'energy', 'maxenergy',
       'class'],
       dtype='object')

- **data.columns** will return you all the column names which are present in your data.

Taking care of Missing Data:

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously, you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values for continuous and for categorical we make use of mode values and replace the missing data.

We will be using **isnull().any()** method to see which column has missing values.

checking for null values

```
In [5]: data.isnull().any()

Out[5]: seismic           False
        seismoacoustic    False
        shift             False
        genergy           False
        gpuls             False
        gdenergy          False
        gdpuls            False
        ghazard           False
        nbumps            False
        nbumps2           False
        nbumps3           False
        nbumps4           False
        nbumps5           False
        nbumps6           False
        nbumps7           False
        nbumps89          False
        energy            False
        maxenergy         False
        class             False
        dtype: bool
```

Since there are no missing values in the dataset, no need to execute this step.

Label encoding

Typically, any structured dataset includes multiple columns with combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too. We need to convert each text category to numbers in order for the machine to process those using mathematical equations.

How should we handle categorical variables? There are Multiple way to handle, but will see one of it is LabelEncoding.

- **Label Encoding** is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

Let's see how to implement label encoding in Python using the scikit-learn library.

As we have to convert only the text class category columns, we first select it then we will implement Label Encoding to it.

```
In [8]: dataset['seismic'].unique()
Out[8]: array(['a', 'b'], dtype=object)

In [9]: dataset['ghazard'].unique()
Out[9]: array(['a', 'b', 'c'], dtype=object)

In [10]: dataset['seismoacoustic'].unique()
Out[10]: array(['a', 'b', 'c'], dtype=object)

In [11]: dataset['shift'].unique()
Out[11]: array(['N', 'W'], dtype=object)

In [12]:
le=LabelEncoder()
dataset['seismic']=le.fit_transform(dataset['seismic'])
dataset['ghazard']=le.fit_transform(dataset['ghazard'])
dataset['seismoacoustic']=le.fit_transform(dataset['seismoacoustic'])
dataset['shift']=le.fit_transform(dataset['shift'])

In [13]: dataset.head(4)
Out[13]:
```

	id	seismic	seismoacoustic	shift	genergy	gpuls	gdenergy	gdpuls	ghazard	nbumps	nbumps2	nbumps3	nbumps4	nbumps5	nbumps6	nbumps7	nb
0	1	0	0	0	15180	48	-72	-72	0	0	0	0	0	0	0	0	
1	2	0	0	0	14720	33	-70	-79	0	1	0	1	0	0	0	0	
2	3	0	0	0	8050	30	-81	-78	0	0	0	0	0	0	0	0	
3	4	0	0	0	28820	171	-23	40	0	1	0	1	0	0	0	0	

Feature Scaling

Splitting Data into Train and Test:

When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.

But the question is, how do you split the data? You can't possibly manually split the dataset into two sets. And you also have to make sure you split the data in a random manner. To help us with this task, the Scikit library provides a tool, called the Model Selection library. There is a class in the library which is, 'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

- **Train Dataset**: Used to fit the machine learning model.

- **Test Dataset**: Used to evaluate the fit machine learning model.

In general you can allocate 80% of the dataset to training set and the remaining 20% to test set.

We will create 4 sets— X_train (training part of the matrix of features), X_test (test part of the matrix of features), Y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices), Y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices.

There are a few other parameters that we need to understand before we use the class:

    test_size — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset

    train_size — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.

    random_state — here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the Random_state class, which will become the number generator. If you don't pass anything, the Random_state instance used by np.random will be used instead.

    Now split our dataset into train set and test using train_test_split class from scikit learn library.

```
In [27]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)

In [28]: print(x_train.shape)
         print(x_test.shape)

         (2067, 24)
         (517, 24)

In [29]: print(y_train.shape)

         (2067, 1)
```

3. Model Building

Training and testing the model:

There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms are Regression algorithms.

Example: 1. Linear Regression.

2. Logistic Regression.

3. Random Forest Regression / Classification.

4. Decision Tree Regression / Classification.

You will need to train the datasets to run smoothly and see an incremental improvement in the prediction rate.

Now we apply Logistic regression algorithm on our dataset.

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In **logistic regression**, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).

LogisticRegression

```
In [33]: lr=LogisticRegression()
         lr.fit(x_train, y_train)
```

```
E:\ana\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
Out[33]: LogisticRegression()
```

```
In [34]: pred=lr.predict(x_test)
         pred
```

```
Out[34]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
```

```
In [35]: r2_scor = r2_score(y_test, pred)
         print("r2_score:",r2_scor)

         accuracy = accuracy_score(y_test, pred)
         print("accuracy(in %):",accuracy*100)
```

```
r2_score: -0.637295582133268
accuracy(in %): 89.92248062015504
```

Now we will apply SVM to our dataset.

Support vector machines (**SVMs**) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of **support vector machines** are: Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples.

SupportVectorMachine

```
In [39]: svm = SVC(kernel = 'sigmoid')
         svm.fit(x_train, y_train)
         pred1=svm.predict(x_test)
```

```
E:\ana\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
In [40]: r2_scor1 = r2_score(y_test, pred1)
         print("r2_score:",r2_scor1)

         accuracy1 = accuracy_score(y_test, pred1)
         print("accuracy(in %):",accuracy1*100)
```

```
r2_score: -0.2594581401025138
accuracy(in %): 92.24806201550388
```

Predict the values

Once the model is trained, it's ready to make predictions. We can use the **predict** method on the model and pass **x_test** as a parameter to get the output as **pred.**

Notice that the prediction output is an array of real numbers corresponding to the input array.

```
In [34]: pred=lr.predict(x_test)
         pred
```

```
Out[34]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
```

Evaluation:

Finally, we need to check to see how well our model is performing on the test data. There are many evaluation techniques are there. For this, we evaluate **r2_score** produced by the model.

```
In [40]: r2_scor1 = r2_score(y_test, pred1)
         print("r2_score:",r2_scor1)

         accuracy1 = accuracy_score(y_test, pred1)
         print("accuracy(in %):",accuracy1*100)

         r2_score: -0.2594581401025138
         accuracy(in %): 92.24806201550388
```

We also plot **Confusion Matrix** for the same to evaluate.

A **confusion matrix** is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.

```
In [36]: from sklearn.metrics import confusion_matrix
         confusion_matrix(y_test, pred)

Out[36]: array([[461,  21],
                [ 31,   3]], dtype=int64)

In [37]: import sklearn.metrics as metrics
         lrsfpr,lrstpr,threshold=metrics.roc_curve(y_test,pred)
         lrsroc_auc=metrics.auc(lrsfpr,lrstpr)

In [38]: import matplotlib.pyplot as plt
         plt.plot(lrsfpr,lrstpr,"b",label="auc=%0.2f"%lrsroc_auc)
         plt.legend(loc='lower right')
         plt.title("roc")
         plt.xlabel("fpr")
         plt.ylabel("tpr")

Out[38]: Text(0, 0.5, 'tpr')
```
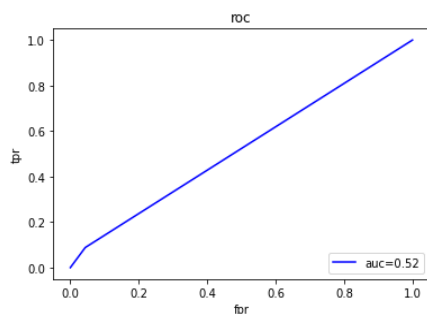

Saving a model:

Model is saved so it can be used in future and no need to train it again.

Pickle

```
In [41]: import pickle
         pickle.dump(lr,open('seismic.pkl','wb'))

In [ ]:
```

4. Application Building

Creating a HTML File, flask application.

Build python code
   a. Importing Libraries

   b. Routing to the html Page

   c. Showcasing prediction on UI

   d. Run The app in local browser
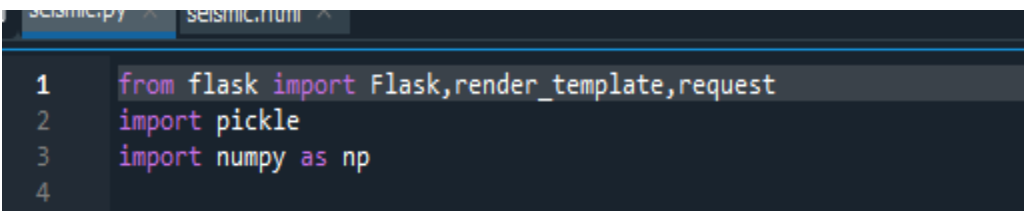
Project Structure:

Create a Project folder that contains files as shown below

Name

📁 templates
📄 dineshwar seismic bumps.ipynb
📄 seismic.pkl
📄 seismic.py
📊 seismic-bumps
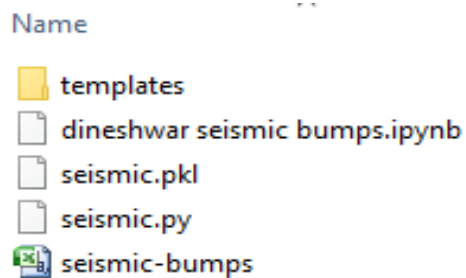
- We are building a Flask Application that needs HTML pages stored in the templates folder

- Templates folder contains seismic.html

- Static folder contains CSS and image files.

Task 1: Importing Libraries

```
1    from flask import Flask,render_template,request
2    import pickle
3    import numpy as np
4
```

Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

Task 2: Routing to the html Page

Here, declared constructor is used to route to the HTML page created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page is rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.
Here, "seismic.html" is rendered when home button is clicked on the UI

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


@app.route('/predict',methods=['post'])
def predict():
    Seismic=float(request.form['seismic'])
    Seismoacoustic=float(request.form['seismoacoustic'])
    Shift=float(request.form['shift'])
    Genergy=float(request.form['genergy'])
    Gpuls=float(request.form['gpuls'])
    Gdenergy=float(request.form['gdenergy'])
    Gdpuls=float(request.form['gdpuls'])
    Ghazard=float(request.form['ghazard'])
    Nbumps=float(request.form['nbumps'])
    Nbumps2=float(request.form['nbumps2'])
    Nbumps3=float(request.form['nbumps3'])
    Nbumps4=float(request.form['nbumps4'])
    Nbumps5=float(request.form['nbumps5'])
    Nbumps6=float(request.form['nbumps6'])
    Nbumps7=float(request.form['nbumps7'])
    Nbumps89=float(request.form['nbumps89'])
    Energy=float(request.form['energy'])
    Maxenergy=float(request.form['maxenergy'])

    a=np.array([[Seismic,Seismoacoustic,Shift,Genergy,Gpuls,Gdenergy,Gdpuls,Ghazard,Nbumps,Nbumps2,Nbumps3,Nbumps4,Nbumps5,Nbumps6,Nbump
    print(a)

    result=lr.predict(a)

    return render_template('seismic.html',x=result)
```

Task 3: Main Function

This is used to run the application in a local host.

```python
if __name__ == '__main__':
    app.run()
```

Activity 3:Run the application

1. Open the anaconda prompt from the start menu.
2. Navigate to the folder where your seismic.py resides.
3. Now type "python app.py" command.
4. It will show the local host where your app is running on **http://127.0.0.1:5000/**
5. Copy that local host URL and open that URL in the browser. It does navigate meto where you can view your web page.

6. Enter the values, click on the predict button and see the result/prediction on the web page.

```
In [2]: runfile('C:/Users/sunkari\'s/Desktop/project ml/seismic.py', wdir='C:/Users/sunkari\'s/Desktop/project ml')
 * Serving Flask app "seismic" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Output Screen: