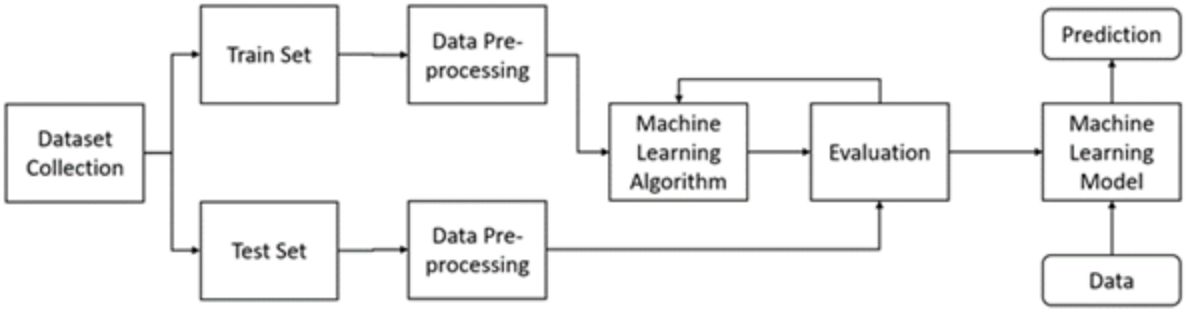
Combined cycle power plant

A **combined cycle power plant** is an assembly of heat engines that work in tandem from the same source of heat, converting it into mechanical energy. On land, when used to make electricity the most common type is called a **combined cycle gas turbine** (**CCGT**) plant. The same principle is also used for marine propulsion, where it is called a combined gas and steam (COGAS) plant. Combining two or more thermodynamic cycles improves overall efficiency, which reduces fuel costs.

The principle is that after completing its cycle in the first engine, the working fluid (the exhaust) is still hot enough that a second subsequent heat engine can extract energy from the heat in the exhaust. Usually the heat passes through a heat exchanger so that the two engines can use different working fluids.

By generating power from multiple streams of work, the overall efficiency of the system can be increased by 50–60%. That is, from an overall efficiency of say 34% (for a simple cycle), to as much as 64% (for a combined cycle). This is more than 84% of the theoretical efficiency of a Carnot cycle. Heat engines can only use part of the energy from their fuel (usually less than 50%), so in a non-combined cycle heat engine, the remaining heat (i.e., hot exhaust gas) from combustion is wasted.

Machine Learning Workflow:



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.

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

Data Collection

The dataset is related to combined cycle power plant. It was taken from the below website:

https://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant

The UCI Learning Repository provides one dataset abalone data, it contains 9568 observations, 4 descriptive features and 1 target feature.

Data Pre-processing

Importing required Libraries:

```
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
        import joblib
        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.

joblib:to serialize your machine learning algorithms and save the serialized format to a file.

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:

```
In [2]: data=pd.read_csv("Folds5x2_pp.csv")

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

Out[3]:

	AT	V	AP	RH	PE
0	8.34	40.77	1010.84	90.01	480.48
1	23.64	58.49	1011.40	74.20	445.75
2	29.74	56.90	1007.15	41.91	438.76
3	19.07	49.69	1007.22	76.79	453.09
4	11.80	40.66	1017.13	97.20	464.43

- 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).

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.

```
In [4]: data.isnull().any()

Out[4]: AT    False
        V     False
        AP    False
        RH    False
        PE    False
        dtype: bool
```

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

Splitting Dataset in to Independent variable and Dependent variable:

To read the columns, we will use iloc of pandas (used to fix the indexes for selection) which takes two parameters — [row selection, column selection].

```
In [13]: x=data1.iloc[:,0:5]
         y=data.iloc[:,1:2]
```

From the above piece of code ":" indicates you are considering all the rows and "0:5" indicates we are considering column 0 to 5 as input values and assigning them to variable x. in the same way in second line ":" indicates you are considering all the rows and "5" indicates we are considering only one column 8 as output value and assigning them to variable y

Feature Scaling:

It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any

variable dominate the other variable.

For feature scaling, we will import ***StandardScaler*** class of ***sklearn.preprocessing***

Now, we will create the object of **StandardScaler** class for independent variables or features. And then we will fit and transform the training dataset.

```
In [26]: from sklearn.preprocessing import StandardScaler
         sc=StandardScaler()

In [27]: x=sc.fit_transform(x)

In [28]: x

Out[28]: array([[-1.51786152, -0.40735691,  1.14394435,  1.53022576],
                [ 0.5352555 , -0.31305658,  0.06103098, -0.50480238],
                [ 1.35381849, -1.02872873, -2.15068773, -0.91438621],
                ...,
                [-0.49130301,  0.15844507,  0.36652077,  0.67941642],
                [-0.26854652,  0.89600837,  1.46176333, -0.20127673],
                [ 0.54062312, -0.2355956 , -0.14171561, -0.15791592]])
```

Model Building

Training and testing the model:

```
In [29]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)

In [31]: x_train.shape,x_test.shape

Out[31]: ((7654, 4), (1914, 4))

In [32]: y_train.shape,y_test.shape

Out[32]: ((7654, 1), (1914, 1))
```

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 Linear regression algorithm on our dataset.

Linear Regression is a **machine learning** algorithm based on supervised **learning**. It performs a **regression** task. **Regression** models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

```
In [33]: from sklearn.linear_model import LinearRegression

         lr=LinearRegression()
         lr.fit(x_train,y_train)
         y_test
```

```
Out[33]: array([[69.13],
                [57.5 ],
                [57.32],
                ...,
                [49.21],
                [60.84],
                [71.64]])
```

- **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]: y_pred_lr=lr.predict(x_test)
         y_pred_lr
```

```
Out[34]: array([[70.65710354],
                [58.13375904],
                [60.09062711],
                ...,
                [53.684039  ],
                [61.29269717],
                [66.82493296]])
```

- 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 [35]: from sklearn.metrics import r2_score
         r2_score(y_test,y_pred_lr)

Out[35]: 0.775772088661744
```

- SAVE MODEL:

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

```
In [36]: import pickle
         pickle.dump(lr,open('ccpp.pkl','wb'))
```

Application Building

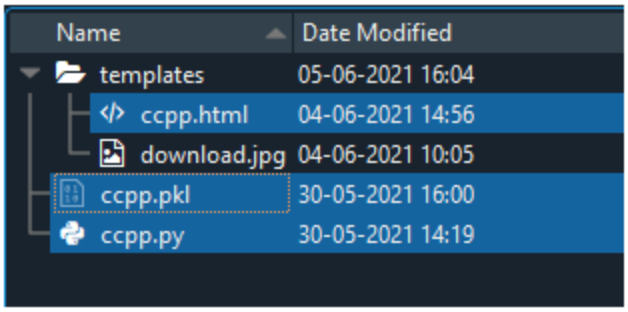
Creating a HTML File, flask application.

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



We are building a Flask Application that needs HTML pages stored in the templates folder

- Templates folder contains index.html
- Static folder contains CSS and image files.

Task 1: Importing Libraries

```
from flask import Flask,render_template,request
import pickle
import numpy as np

app=Flask(__name__) #your application
lr=pickle.load(open('ccpp.pkl','rb'))
```

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, "ccpp.html" is rendered when home button is clicked on the UI

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


@app.route('/predict',methods=['post'])
def predict():
    AT=float(request.form['AT'])
    AP=float(request.form['AP'])
    RH=float(request.form['RH'])
    PE=float(request.form['PE'])

    print(AT,AP,RH,PE)
    a=np.array([[AT,AP,RH,PE]])

    result=lr.predict(a)

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

Task 3: Main Function

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

```
if __name__ == '__main__':
    app.run(port=8000) # you are running your app
```

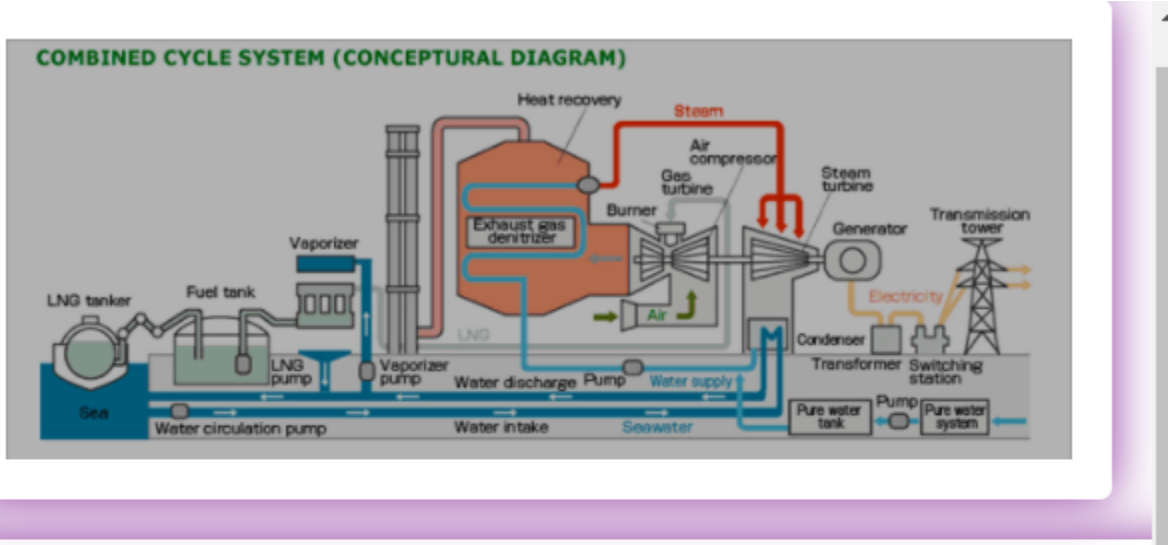
Activity 3:Run the application

Open the spyder from the start menu.
Navigate to the folder where your ccpp.py resides.
Now open the “ccpp.py” file in spyder
It will show the local host where your app is running on **http://127.0.0.1:8000/**
Copy that local host URL and open that URL in the browser. It does navigate meto where you can view your web page.
Enter the values, click on the predict button and see the result/prediction on the web page.

```
IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Ganesh/OneDrive/Desktop/Flask/ccpp.py', wdir='C:/Users/Ganesh/
OneDrive/Desktop/Flask')
 * Serving Flask app "ccpp" (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://127.0.0.1:8000/ (Press CTRL+C to quit)
```

Output Screen:



Enter AT

8.34

Enter AP

1010.84

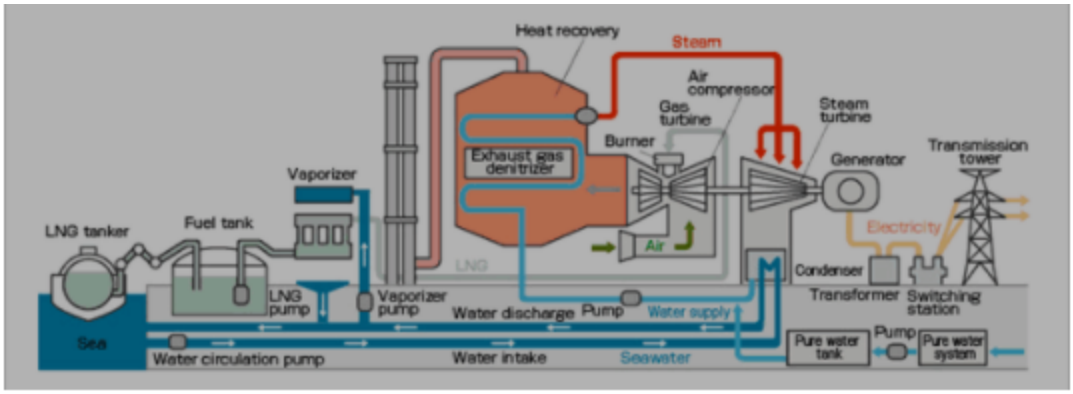
Enter RH

90.01

Enter PE

480.48

click



Enter AT

Enter AP

Enter RH

Enter PE

click

[[54.62895187]]