Architecture:

Prediction of full load electrical power output of a base load operated combined cycle power plant using Machine Learning

---

System Required:

Windows 8 machine Install with two web browser Bandwidth of 30mbps

Project Description

Our project focuses on predicting the full load electrical power output of a base load operated combined cycle power plant using machine learning. By analyzing operational data, ambient conditions, fuel characteristics, and equipment status, we develop a predictive model. This model assists power plant operators in optimizing performance, scheduling maintenance, and managing grid stability by accurately forecasting power output under varying conditions.

Real-Time Scenarios:

Performance Optimization

Operators use the model to predict power output under different operating conditions, optimizing plant performance and maximizing efficiency while meeting demand requirements.

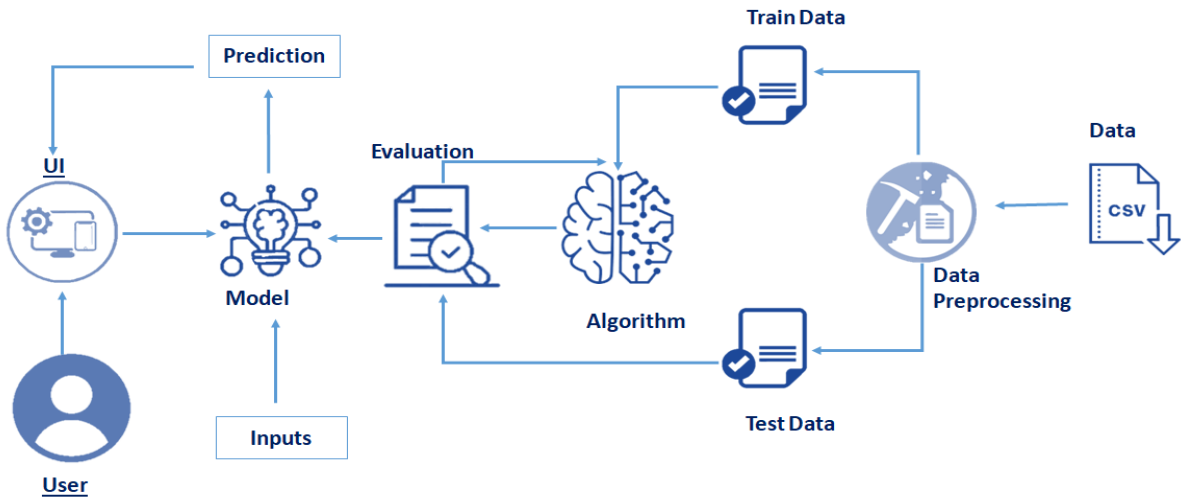
Maintenance Scheduling

Maintenance teams utilize the model to forecast power output trends, scheduling maintenance activities during periods of anticipated lower demand to minimize downtime and revenue loss.

Grid Stability Management

Grid operators leverage the model to anticipate fluctuations in power output from the plant, enabling proactive measures to maintain grid stability and reliability, such as adjusting reserve capacities and load shedding strategies. This ensures consistent power supply to consumers and prevents disruptions in the electrical grid during peak demand periods or unforeseen operational challenges.

Architecture:



Learning Outcomes

By the end of this project:

- You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
- You will be able to know how to pre-process / clean the data using different data pre-processing techniques.
- You will be able to analyze or get insights of data through visualization.
- Applying different algorithms according to dataset and based on visualization.
- You will be able to know how to find the accuracy of the model.
- You will be able to know how to build a web application using the Flask framework.

Prerequisites

To complete this project, you must require following software's, concepts and packages

Software Installations

In order 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

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video.

Architecture:

Packages Installation

To build Machine learning models you must require the following packages

- **Numpy:**
- It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations
- **Scikit-learn:**
    - It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- **Matplotlib and Seaborn:**
    - Matplotlib is mainly deployed for basic plotting. Visualization using Matplotlib generally consists of bars, pies, lines, scatter plots and so on. Seaborn: Seaborn, on the other hand, provides a variety of visualization patterns. It uses fewer syntax and has easily interesting default themes.
- **Pandas:**
    - It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

- **Pickle:** The pickle module implements serialization protocol, which provides an ability to save and later load Python objects using special binary format.

If you are using **anaconda navigator**, follow below steps to download required packages:

- Open the anaconda prompt.
- Type “pip install jupyter notebook” and click enter.
- Type “pip install spyder” and click enter.
- Type “pip install numpy” and click enter.
- Type “pip install pandas” and click enter.
- Type “pip install matplotlib” and click enter.
- Type “pip install seaborn” and click enter.
- Type “pip install sklearn” and click enter.
- Type “pip install Flask” and click enter.

Architecture:

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Prior Knowledge

One should have knowledge on the following Concepts:

Watch the below video to know about the types of machine learning

It is recommended to watch the above video's to understand the concepts before you start your project.

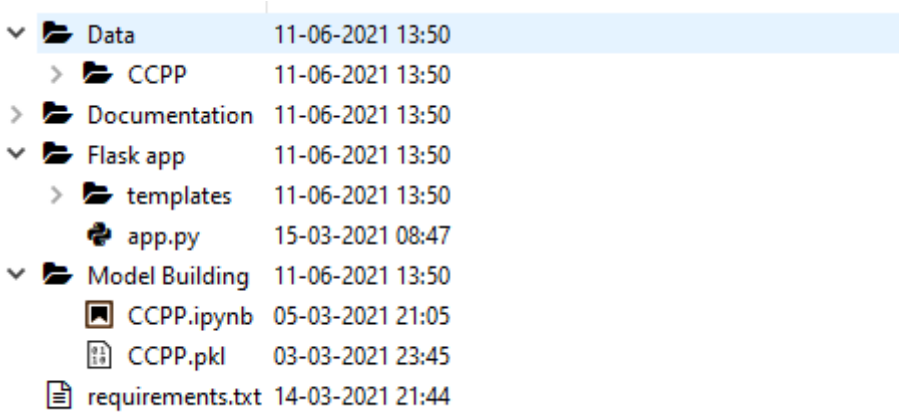
Project Workflow

- User interacts with the UI (User Interface) to upload the input features.
- Uploaded features/input is analyzed by the model which is integrated.
- Once a model analyses the uploaded inputs, the prediction is showcased on the UI.

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

Project Structure

Create a Project folder which contains files as shown below.



- We are building a Flask Application which needs HTML pages stored in the templates folder and a python script CCPP_Flask_App.ipynb for serverside scripting
- CCPP_Flask_App.ipynb - contains the actual python code that will import the app and start the development server.
- CCPP.ipynb - This is where you define models for your application.
- CCPP.pkl - This is our model weights file

Architecture:

- templates - This is where you store your html templates i.e. index.html, home.html, predict.html

Dataset Collection

- ML depends heavily on data, without data, it is impossible for an "AI" to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.
  Importing the Libraries
- The first step is usually importing the libraries that will be needed in the program.

- The required libraries to be imported to Python script are:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
```

Collect the dataset or create the dataset

- The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load. Features consist of hourly average ambient variables Temperature (T), Ambient Pressure (AP), Relative Humidity (RH) and Exhaust Vacuum (V) to predict the net hourly electrical energy output (EP) of the plant.
- A combined cycle power plant (CCPP) is composed of gas turbines (GT), steam turbines (ST) and heat recovery steam generators. In a CCPP, the electricity is generated by gas and steam turbines, which are combined in one cycle, and is transferred from one turbine to another. While the Vacuum is collected from and has effect on the Steam Turbine, he other three of the ambient variables effect the GT performance.
- You can collect dataset from different open sources like kaggle.com, data.gov, UCI machine learning repository etc

- Numpy: It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.
- Pandas: It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- Matplotlib: Visualisation with python. It is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Seaborn: Seaborn is a library for making statistical graphics in Python. Seaborn helps you explore and understand your data. Its plotting functions operate on dataFrames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.
- Pickle: The pickle module implements serialization protocol, which provides an ability to save and later load Python objects using special binary format.

- **Reading the Dataset**

Architecture:

- 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).
-
- names on Windows tend to have backslashes in them. But we want them to mean actual backslashes, not special characters.
-

```
# import the dataset from specified location
data=pd.read_csv('E:/Datascience/Datasets/CCPP/csv/Folds5x2_pp.csv')
```

- If the dataset is in the same directory of your program, you can directly read it, without giving raw as r.
- In this dataset Features consist of hourly average ambient variables
- Temperature (T) in the range 1.81°C and 37.11°C,
- Ambient Pressure (AP) in the range 992.89-1033.30 milibar,
- Relative Humidity (RH) in the range 25.56% to 100.16%
- Exhaust Vacuum (V) in teh range 25.36-81.56 cm Hg
- Net hourly electrical energy output (EP) 420.26-495.76 MW
- The averages are taken from various sensors located around the plant that record the ambient variables every second. The variables are given without normalization.
-

Exploratory Data Analysis

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

```
# showing the data from top 5
data.head()
```

	AT	V	AP	RH	PE
0	14.96	41.76	1024.07	73.17	463.26
1	25.18	62.96	1020.04	59.08	444.37
2	5.11	39.40	1012.16	92.14	488.56
3	20.86	57.32	1010.24	76.64	446.48
4	10.82	37.50	1009.23	96.62	473.90

-

- Head() method is used to return top n (5 by default) rows of a DataFrame or series.
- **Tail():** To check the last five rows of the dataset, we have a function called **tail().**

Architecture:

```
# showing the data from bottom 5
data.tail()
```

	AT	V	AP	RH	PE
9563	16.65	49.69	1014.01	91.00	460.03
9564	13.19	39.18	1023.67	66.78	469.62
9565	31.32	74.33	1012.92	36.48	429.57
9566	24.48	69.45	1013.86	62.39	435.74
9567	21.60	62.52	1017.23	67.87	453.28

- **Understanding Data Type and Summary of features**
- How the information is stored in a DataFrame or Python object affects what we can do with it and the outputs of calculations as well. There are two main types of data those are numeric and text data types.
- Numeric data types include integers and floats.
- Text data type is known as Strings in Python, or Objects in Pandas. Strings can contain numbers and / or characters.
- or example, a string might be a word, a sentence, or several sentences.
- Will see how our dataset is, by using the info() method.

```
# Print a concise summary of a DataFrame.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9568 entries, 0 to 9567
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AT      9568 non-null   float64
 1   V       9568 non-null   float64
 2   AP      9568 non-null   float64
 3   RH      9568 non-null   float64
 4   PE      9568 non-null   float64
dtypes: float64(5)
memory usage: 373.9 KB
```

- We notice all the columns in dataset is float64 bit characters. There is no categorical data present in our dataset. But it is not necessary that all the continuous data which we are seeing has to be continuous in nature. There may be a case that some categorical data is in the form of numbers but when we perform info() operation we will get numerical output. So, we need to take care of those

Architecture:

type of data also.

- **describe():** functions are used to compute values like count, mean, standard deviation and IQR(Inter Quantile Ranges) and give a summary of numeric type data.

```
# Computes a summary of statistics pertaining to the DataFrame columns
data.describe()
```

	AT	V	AP	RH	PE
count	9568.000000	9568.000000	9568.000000	9568.000000	9568.000000
mean	19.651231	54.305804	1013.259078	73.308978	454.365009
std	7.452473	12.707893	5.938784	14.600269	17.066995
min	1.810000	25.360000	992.890000	25.560000	420.260000
25%	13.510000	41.740000	1009.100000	63.327500	439.750000
50%	20.345000	52.080000	1012.940000	74.975000	451.550000
75%	25.720000	66.540000	1017.260000	84.830000	468.430000
max	37.110000	81.560000	1033.300000	100.160000	495.760000

-
- **Checking for null values**
- 1. After loading it is important to check the complete information of data as it can indication many of the hidden information such as null values in a column or a row
- 2.Check whether any null values are there or not. if it is present then following can be done,
- a.Imputing data using Imputation method in sklearn
- b.Filling NaN values with mean, median and mode using fillna() method.
- We will be using isnull().sum() method to see which column has missing values by counting the total sum of null values in each column.

```
# It returns the number of
# missing values in the data set
data.isnull().sum()
```

```
AT    0
V     0
AP    0
RH    0
PE    0
dtype: int64
```

-
- As our data is not having any null values, we can proceed further.
-

Data Visualization

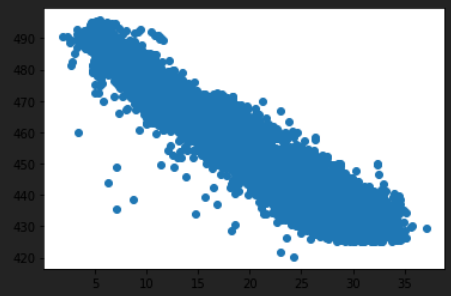
- Data visualization is where a given data set is presented in a graphical format. It helps the detection of patterns, trends and correlations that might go undetected in text-based data.
- Understanding your data and the relationship present within it is just as important as any algorithm used to train your machine learning model. In fact, even the most sophisticated machine learning models will perform poorly on data that wasn't visualized and understood properly.
- To visualize the dataset we need libraries called Matplotlib and Seaborn.
- The Matplotlib library is a Python 2D plotting library which allows you to generate plots, scatter plots, histograms, bar charts etc.
- Univariate Analysis
- Univariate analysis is the simplest form of data analysis where the data being analyzed contains only one variable.
- Bivariate Analysis

Architecture:

- It involves the analysis of two variables (often denoted as X, Y), for the purpose of determining the empirical relationship between them.
- Let’s visualize our data using Matplotlib and searborn library.
- Temperature (T)
- Ambient Pressure (AP)
- Relative Humidity (RH)
- Exhaust Vacuum (V)
- Net hourly electrical energy output (EP or PE)
- Plotting scatter plot to summarize "AI" and "PE" data.

```
# draw a scatter plot diagram
# The x-axis represents AT, and the y-axis represents PE.
plt.scatter(data['AT'],data['PE'])
```

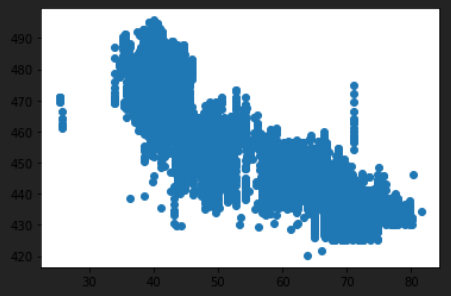
```
<matplotlib.collections.PathCollection at 0x1c612733b80>
```



- 2. Plotting scatter plot to summarize "V" and "PE" data.

```
# draw a scatter plot diagram
# The x-axis represents V, and the y-axis represents PE.
plt.scatter(data['V'],data['PE'])
```

```
<matplotlib.collections.PathCollection at 0x1c6127f1370>
```

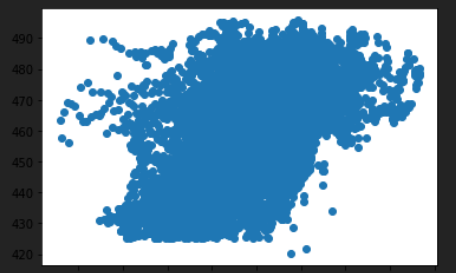


- 3. Plotting scatter plot to summarize "AP" and "PE" data.

Architecture:

```
# draw a scatter plot diagram
# The x-axis represents AP, and the y-axis represents PE.
plt.scatter(data['AP'],data['PE'])
```

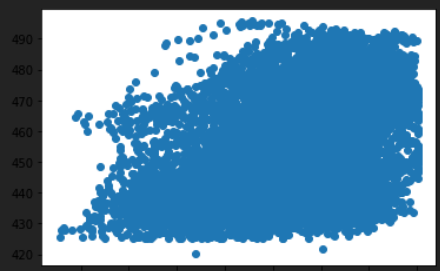
```
<matplotlib.collections.PathCollection at 0x1c61290c370>
```



- 4. Plotting scatter plot to summarize "RH" and "PE" data

```
# draw a scatter plot diagram
# The x-axis represents RH, and the y-axis represents PE.
plt.scatter(data['RH'],data['PE'])
```

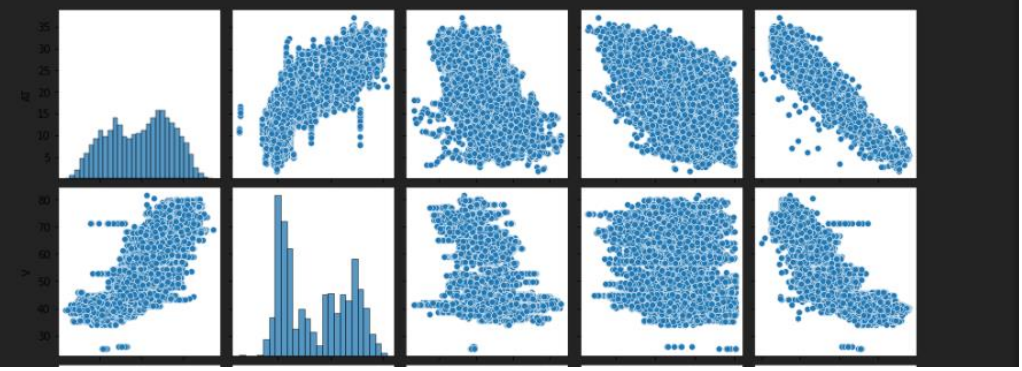
```
<matplotlib.collections.PathCollection at 0x1c612aed4c0>
```



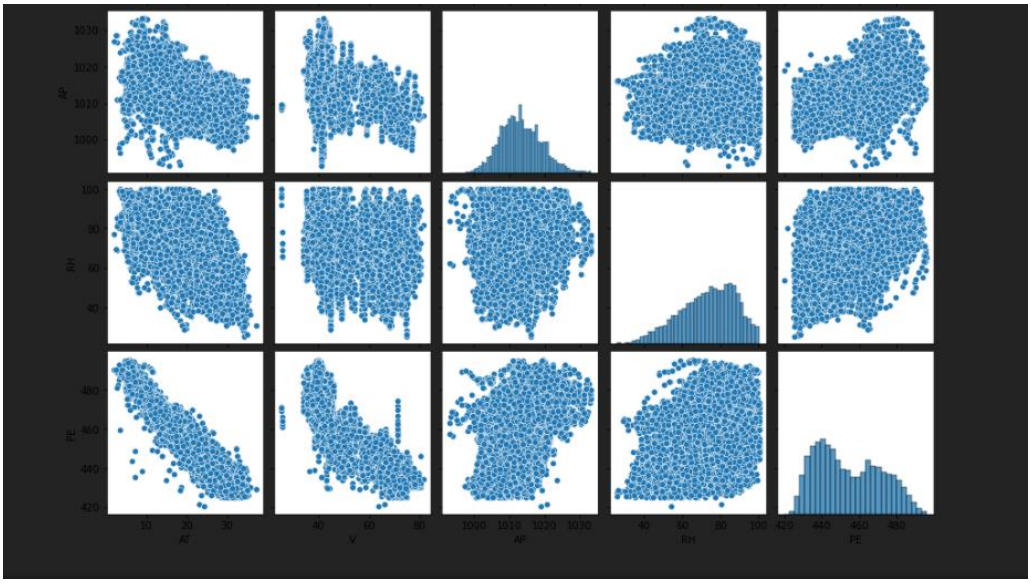
- 5. Plot Pair Plot between all columns

```
# The pairplot function creates a grid of Axes such that each variable
# in data will by shared in the y-axis across a single row and in the x-axis across a single column.
sns.pairplot(data, diag_kind = 'hist')
```

```
<seaborn.axisgrid.PairGrid at 0x1c6068f9e50>
```



Architecture:



- **Splitting the Dataset into Dependent and Independent variable**
- In machine learning, the concept of dependent variable (y) and independent variables(x) is important to understand. Here, Dependent variable is nothing but output in dataset and independent variable is all inputs in the dataset.
- With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, Vector is defined as a matrix that has just one column.
- 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].
- Let's split our dataset into independent and dependent variables.
- 1. The independent variable in the dataset would be considered as 'x'.
- 2. The dependent variable in the dataset would be considered as 'y' and the 'PE' column is considered as dependent variable.
- Now we will split the data of independent variables.

```
# dividing the data into input and output
x=data.drop(['PE'],axis=1)
y=data['PE']
```

- In the above code we are creating array or list of the independent variable x with our selected columns and for dependent variable y we are only taking the PE column.
- **Split the Dependent and Independent Features into Train set and Test set**
- 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.

Architecture:

- 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.

```
# importing the train_test_split from scikit-learn
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

- Check the shape of both xtrain and xtest.

```
# Returns size of xtrain
xtrain.shape
```

```
(7654, 4)
```

```
# Returns size of xtest
xtest.shape
```

```
(1914, 4)
```

- We notice that, both xtest and xtrain contains the 4 columns.

Architecture:

Data Pre-processing

Data Pre-processing includes the following main tasks

1. Import the Libraries.
2. Reading the dataset.
3. Analyse the data.
4. Taking Care of Missing data.
5. Data Visualization.
6. Splitting the Dataset into Dependent and Independent variables.
7. Splitting Data into Train and Test

Model Building

Predictive modeling is a mathematical approach to create a statistical model to forecast future behavior based on input test data.

Steps involved in predictive modeling:

Algorithm Selection:

When we have the structured dataset, and we want to estimate the continuous or categorical outcome then we use supervised machine learning methodologies like regression and classification techniques. When we have unstructured data and want to predict the clusters of items to which a particular input test sample belongs, we use unsupervised algorithms. An actual data scientist applies multiple algorithms to get a more accurate model.

Train Model:

After assigning the algorithm and getting the data handy, we train our model using the input data applying the preferred algorithm. It is an action to determine the correspondence between independent variables, and the prediction targets.

Model Prediction:

Architecture:

We make predictions by giving the input test data to the trained model. We measure the accuracy by using a cross-validation strategy or ROC curve which performs well to derive model output for test data.

Model building includes the following main tasks

1. Training and testing the model
2. Evaluation of Model
3. Save the model
4. Predicting the output using the model

Train and Test the Model using Regression Algorithms

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 may be Classification algorithms are Regression algorithms.

Example:

1. Decision Tree Regression / Classification.

2. Random Forest Regression / Classification.

3. Linear Regression.

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

Now we apply regression algorithms on our dataset.

Decision Tree Regression: A decision tree is a supervised machine learning model used to predict a target by learning decision rules from features.

Random Forest Regression: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees.

Multiple Linear Regression: In statistics, Multiple linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables).

Build the model

We're going to use x_train and y_train obtained above in train_test_split section to train our regression model. We're using the fit method and passing the parameters as shown below.

1. Import the Regression algorithms

Architecture:

```
# Importin the machine learning models
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

2. Initialize the model

```
# Initializing the models
LRmodel=LinearRegression()
DTmodel=DecisionTreeRegressor()
RFmodel=RandomForestRegressor()
```

3. Training model with our data.

- **Linear Model**

```
# Linear Regression
from sklearn.linear_model import LinearRegression
# Initializing the model
LRmodel = LinearRegression()

# Train the data with Linear Regreesion model
LRmodel.fit(xtrain, ytrain)
```

```
LinearRegression()
```

- **Decision Tree Regression**

```
# Decision Tree regressor
from sklearn.tree import DecisionTreeRegressor
# Intializing the model
DTRmodel=DecisionTreeRegressor()

# Train the data with Linear Regreesion model
DTRmodel.fit(xtrain, ytrain)
```

```
DecisionTreeRegressor()
```

Architecture:

Random Forest Regression

```
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
# Initializing the model
RFmodel=RandomForestRegressor()
```

```
# Train the data with Random Forest model
RFmodel.fit(xtrain, ytrain)
```

```
RandomForestRegressor()
```

Model evaluation

Finally, we need to check to see how well our model is performing on the test data.

Evaluation Metrics:

- r2_score of Linear Regressor is

Architecture:

```
# Linear Regression
from sklearn.linear_model import LinearRegression
# Initializing the model
LRmodel = LinearRegression()

# Train the data with Linear Regreesion model
LRmodel.fit(xtrain, ytrain)

 LinearRegression()

LRpred=LRmodel.predict(xtest)

# Importing R Square library
from sklearn.metrics import r2_score

# Checking for accuracy score with actual data and predicted data
LRscore=r2_score(ytest, LRpred)
LRscore

 0.9325315554761302
```

- r2_score of Decision Tree Regressor is

```
# Decision Tree regressor
from sklearn.tree import DecisionTreeRegressor
# Intializing the model
DTRmodel=DecisionTreeRegressor()

# Train the data with Linear Regreesion model
DTRmodel.fit(xtrain, ytrain)

 DecisionTreeRegressor()

DTRpred=DTRmodel.predict(xtest)

# Checking for accuracy score with actual data and predicted data
DTRscore=r2_score(ytest, DTRpred)
DTRscore

 0.9224221478563223
```

- r2_score of Random Forest Regressor is

Architecture:

```
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
# Initializing the model
RFmodel=RandomForestRegressor()
```

```
# Train the data with Random Forest model
RFmodel.fit(xtrain, ytrain)
```

```
RandomForestRegressor()
```

```
RFpred=RFmodel.predict(xtest)
```

```
# Checking for accuracy score with actual data and predicted data
RFscore=r2_score(ytest, RFpred)
RFscore
```

```
0.9647395938141367
```

As we can see that the r2_score of the Decision Tree Regressor and linear regressor is less than the random forest regressor model, we are proceeding with the Random Forest Regressor model.

Save the Model

After building the model we have to save the model.

Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.

This is done by the below code

```
# saving the model
pickle.dump(RFmodel, open('CCPP.pkl','wb'))
```

Here, RFmodel is our Random Forest Regression class, saving as CCPP.pkl file. Wb is the write binary in bytes.

Application Building

Application Building involves following steps

Architecture:

1. Create an HTML file
2. Build a Python Code
3. Run the app

Create an HTML File

- We use HTML to create the front-end part of the web page.
- Here, we created 2 html pages- home.html, index.html.
- home.html displays the home page.
- index.html accepts the values from the user and displays the prediction.
- For more information regarding HTML refer the link below
    - Link
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

- Link 1 :
- Link 2

Build Python Code

Let us build flask file 'Liver_Flask_App.py' which is a web framework written in python for server-side scripting.

Let's see step by step procedure for building the backend application.

- App starts running when "__name__" constructor is called in main.
- render_template is used to return html file.
- "GET" method is used to take input from the user.
- "POST" method is used to display the output to the user.

Importing Libraries

```
from flask import Flask, render_template, request # Flask is a application
# used to run/serve our application
# request is used to access the file which is uploaded by the user in out application
# render_template is used for rendering the html pages
import pickle # pickle is used for serializing and de-serializing Python object structures
```

Architecture:

Libraries required for the app to run are to be imported.

Creating our flask app and loading the model

```
app=Flask(__name__) # our flask app
```

Now after all the libraries are imported, we will be creating our flask app. and the load our model into our flask app.

Routing to the html Page:

@app.route is used to route the application where it should route to.

'/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page is rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Here, "home.html" is rendered when the home button is clicked on the UI and "index.html" is rendered when the predict button is clicked.

```
@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
@app.route('/predict') # rendering the html template
def index() :
    return render_template("index.html")
```

Firstly, we are rendering the home.html template and from there we are navigating to our prediction page that is upload.html. We enter input values here and these values are sent to the loaded model and the resultant output is displayed on index.html.

Architecture:

```
@app.route('/data_predict', methods=['POST']) # route for our prediction
def predict():
    at = request.form['at'] # requesting for age data
    v = request.form['v'] # requesting for gender data
    ap = request.form['ap'] # requesting for Total_Bilirubin data
    rh = request.form['rh'] # requesting for Direct_Bilirubin data

    # coverting data into float format
    data = [[float(at), float(v), float(ap), float(rh)]]

    # loading model which we saved
    model = pickle.load(open('CCPP.pkl', 'rb'))

    prediction= model.predict(data)[0]
    return render_template('predict.html', prediction=prediction)

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

Here the route for prediction is given and necessary steps are performed in order to get the predicted output.

Lastly, we run our app on the local host. Here we are running it on localhost:5000

```
@app.route('/data_predict', methods=['POST']) # route for our prediction
def predict():
    at = request.form['at'] # requesting for age data
    v = request.form['v'] # requesting for gender data
    ap = request.form['ap'] # requesting for Total_Bilirubin data
    rh = request.form['rh'] # requesting for Direct_Bilirubin data

    # coverting data into float format
    data = [[float(at), float(v), float(ap), float(rh)]]

    # loading model which we saved
    model = pickle.load(open('CCPP.pkl', 'rb'))

    prediction= model.predict(data)[0]
    return render_template('predict.html', prediction=prediction)

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

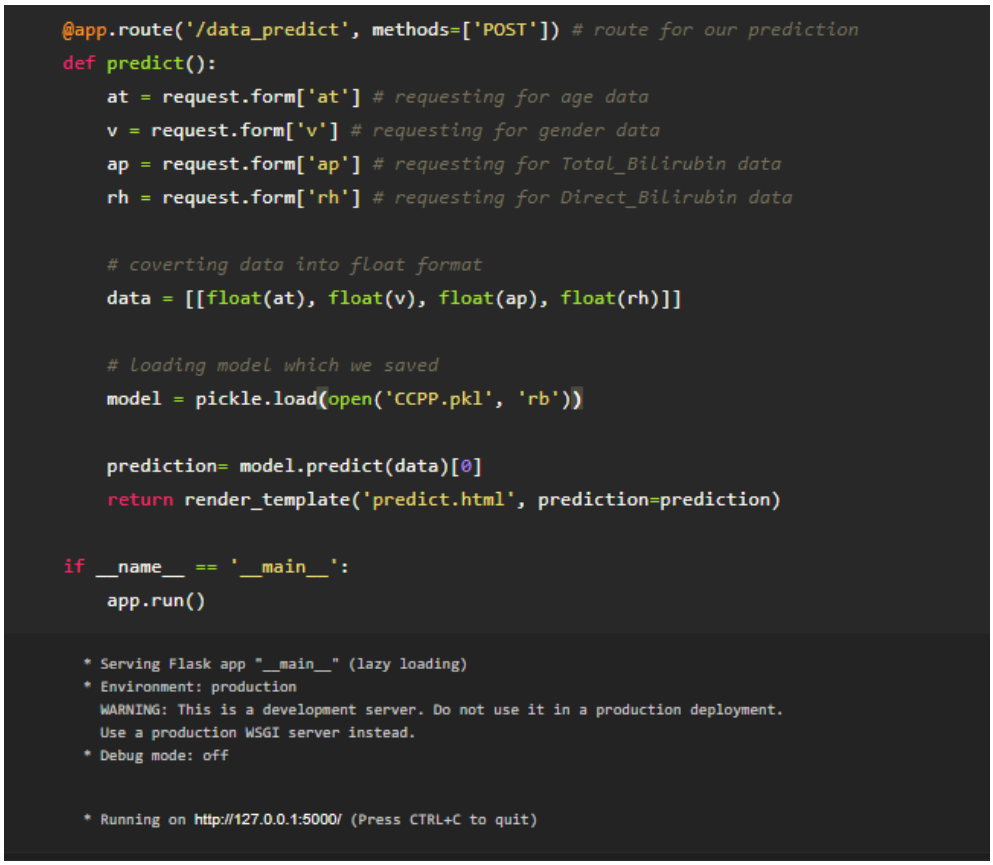
```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off


 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Architecture:

Run the App

Run the application from anaconda prompt

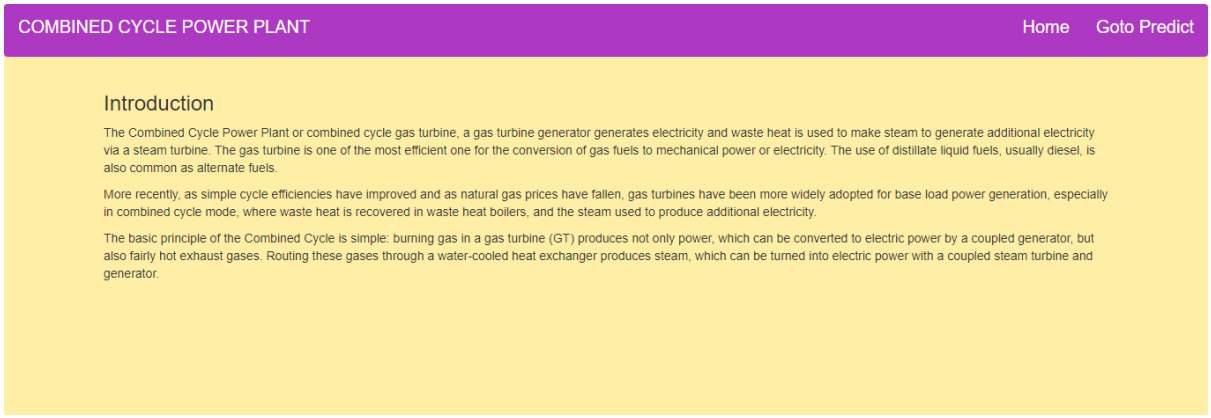
- Open new anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- It will show the local host where your app is running on **http://127.0.0.1.8000/**
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.



```
@app.route('/data_predict', methods=['POST']) # route for our prediction
def predict():
    at = request.form['at'] # requesting for age data
    v = request.form['v'] # requesting for gender data
    ap = request.form['ap'] # requesting for Total_Bilirubin data
    rh = request.form['rh'] # requesting for Direct_Bilirubin data

    # coverting data into float format
    data = [[float(at), float(v), float(ap), float(rh)]]

    # loading model which we saved
    model = pickle.load(open('CCPP.pkl', 'rb'))

    prediction= model.predict(data)[0]
    return render_template('predict.html', prediction=prediction)

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

```
 * Serving Flask app "__main__" (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)
```

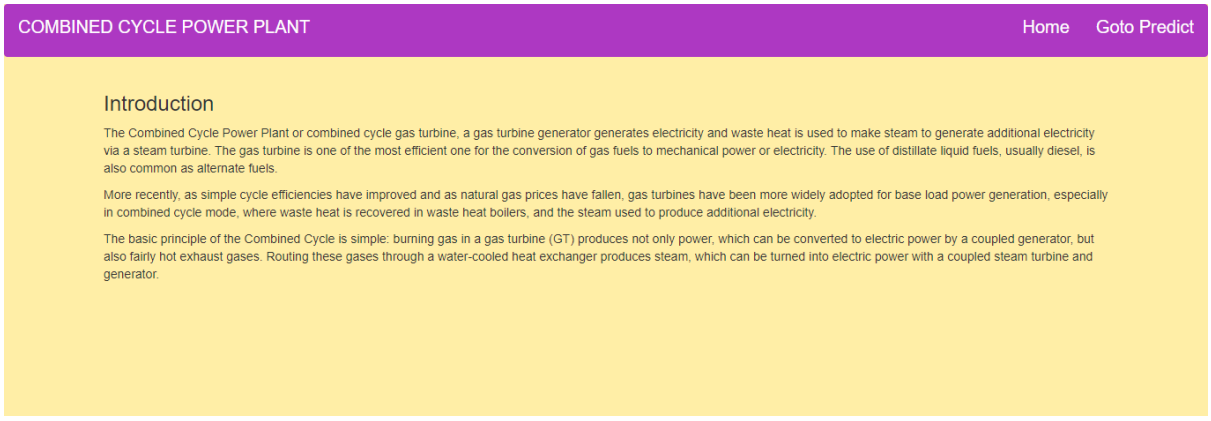
Architecture:



- Showcasing the output on UI

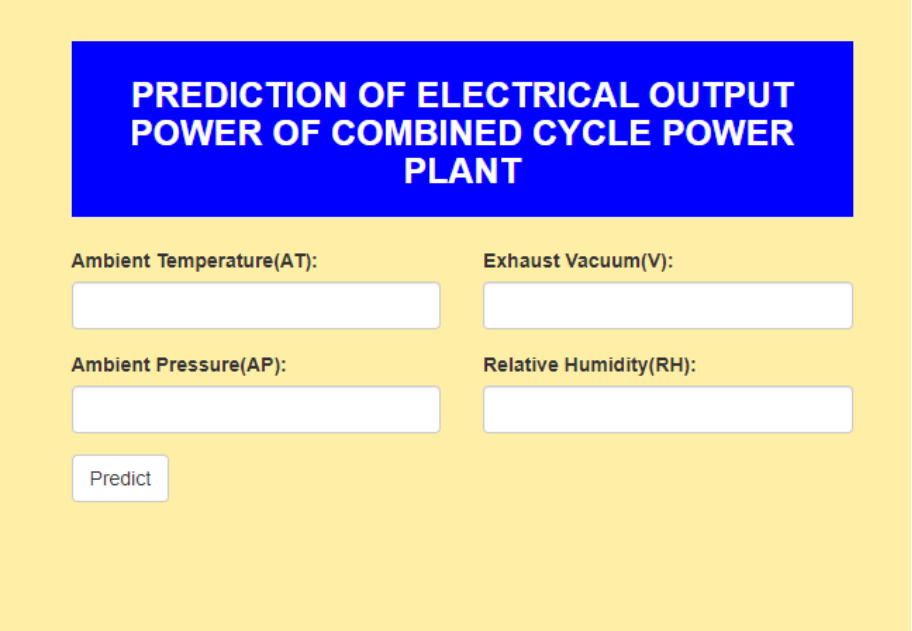
Home page is displayed when home button is clicked. Predict page is displayed when predict button is clicked. In predict page, enter input values to predict the power out from ccpp. Finally, the prediction for the given input features is shown.

Home Page:

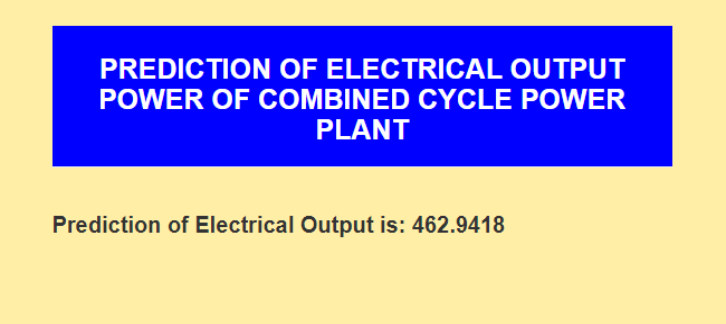


Prediction Page:

Architecture:



Output:



As we see the predicted output is displayed on the User Interface

Thank you