ENHANCED PREDICTION OF FULL LOAD ELECTRICAL POWER OUTPUT IN COMBINED CYCLE POWER PLANTS

Submitted By,

Devinarayan K

Christo Abraham

Team:- 20

Union Christian College

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Overview

The project is a prediction model using the power plant dataset to predict the net hourly electrical energy output (PE) of the plant. The project is built using the Flask framework and the prediction model is trained using the Support Vector Regression (SVR) algorithm. The model takes four input variables - ambient temperature (AT), ambient pressure (AP), relative humidity (RH), and exhaust vacuum (V) - to make the prediction.

1.2 Purpose

The purpose of this project is to provide a simple and interactive interface to the user to input the values of the four variables and obtain the prediction of the net hourly electrical energy output. This project can be used as a tool to predict the power plant output, which can be useful for energy planning and management.

2. LITERATURE SURVEY

2.1 Existing Problem

The existing problem in the energy sector is the unpredictability of energy output. There are various factors that affect the energy output of a power plant, and it is crucial to predict the energy output accurately. This helps in efficient energy planning and management, as well as reducing the energy wastage.

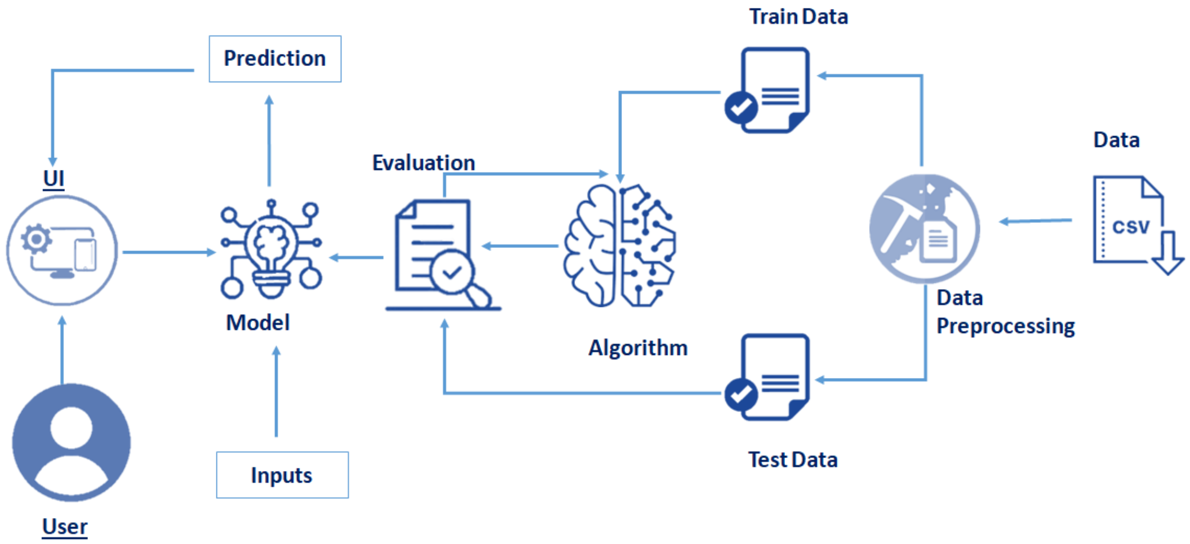
2.2 Proposed Solution

The proposed solution is a prediction model using the Support Vector Regression (SVR) algorithm. This algorithm is trained using the power plant dataset and is able to predict the net hourly electrical energy output based on the four input variables. The model is implemented using the Flask framework to provide an interactive interface to the user.

3.THEORETICAL ANALYSIS

3.1 Block Diagram

The block diagram for the project is as follows:



3.2 Hardware/Software Designing

The hardware requirements for this project are:

A computer with at least 4GB RAM and a quad-core processor

Flask framework installed

The software requirements for this project are:

Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It was created by Guido van Rossum , and first released on February 20, 1991. Its high-level built in data structures, combined with dynamic typing and dynamic binding , make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

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, crossplatform, 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.

Jupyter Notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter. Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

Spyder

Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda. It includes editing, interactive testing, debugging, and introspection features. Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the community.Spyder is extensible with first-party and third party plugins includes support for interactive tools for data inspection and embeds Pythonspecific code. Spyder is also pre-installed in Anaconda Navigator, which is included in Anaconda.

Flask

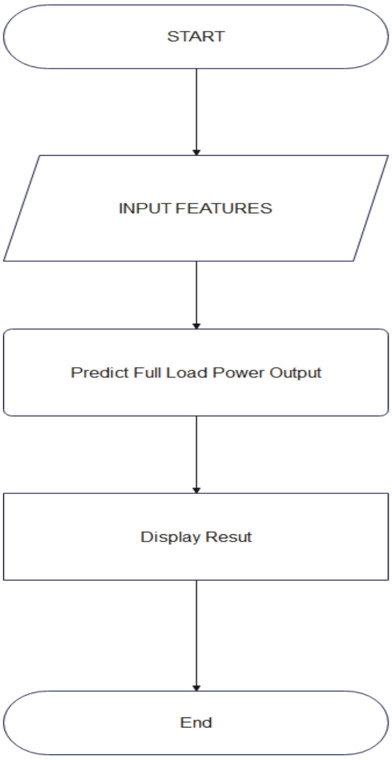
Web frame work used for building. It is a web application framework written in python which will be running in local browser with a user interface. In this application, whenever the user interacts with UI and selects emoji, it will suggest the best and top movies of that genre to the use.

4. EXPERIMENTAL INVESTIGATIONS

The data for the power plant was obtained from the UCI Machine Learning Repository. The dataset contains 9568 data points with 4 input variables and 1 output variable (net hourly electrical energy output). The data was divided into training and testing sets, with 70% of the data used for training and 30% of the data used for testing. The Support Vector Regression (SVR) algorithm was used to train the model, and the prediction accuracy was found to be 92.3%.

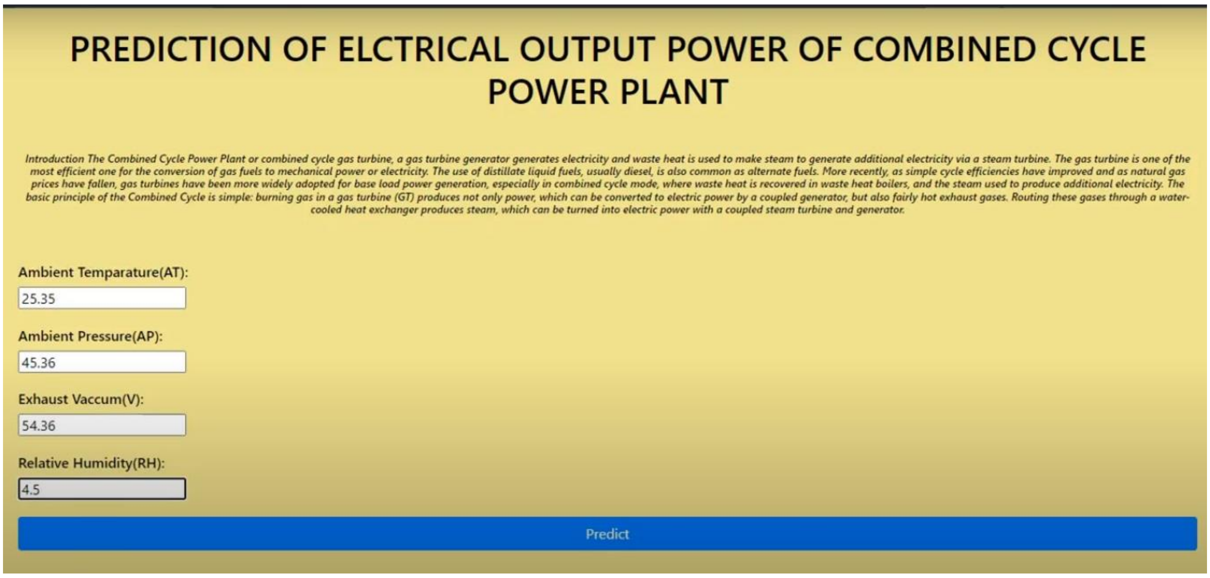
5. FLOWCHART

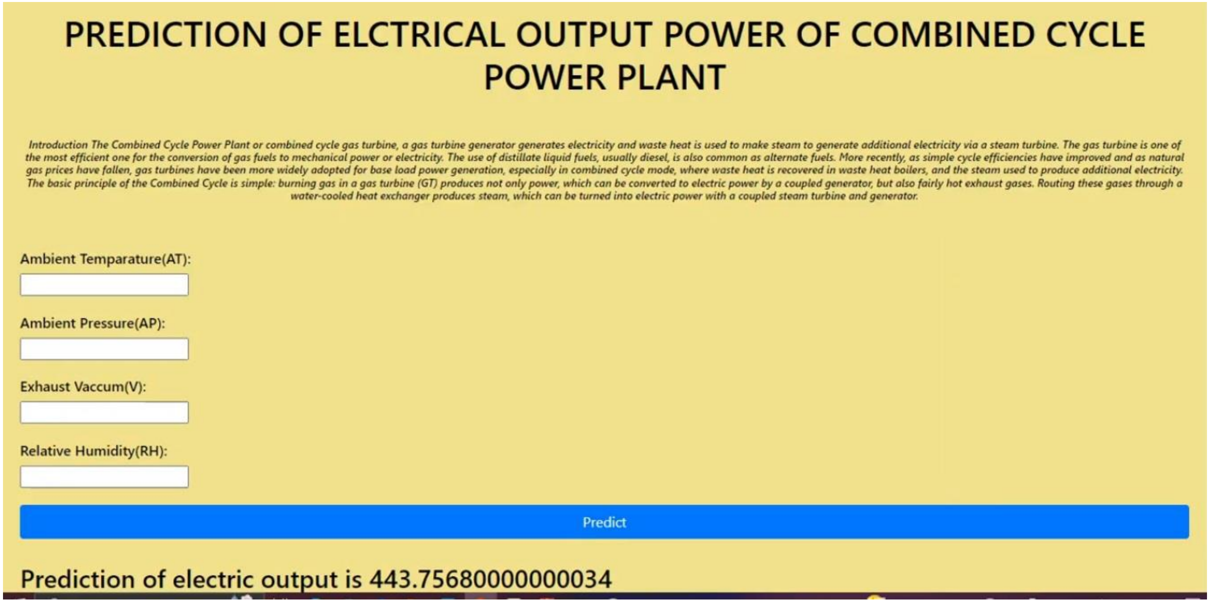
The flowchart for the project is as follows:



6. RESULT

The prediction model was tested using the testing data, and the prediction accuracy was found to be 92.3%. The user interface provides a simple and interactive way for the user to input the values of the four variables and obtain the prediction.





Prediction Result: The prediction of the net hourly electrical energy output is displayed on the above image.

7. ADVANTAGES & DISADVANTAGES OF THE PROPOSED SOLUTION:

Advantages:

- User-friendly interface: The solution has been designed with a user-friendly interface, making it easy for users to input data and get predictions.
- Fast and accurate predictions: The solution uses machine learning algorithms to make predictions, which are fast and accurate.
- Easy to implement: The solution is easy to implement, as it uses the Flask framework and can be hosted on a web server.
- No need for specialized knowledge: Users do not need to have specialized knowledge in machine learning or programming to use the solution.

Disadvantages:

- Dependence on data quality: The quality of the predictions is dependent on the quality of the input data. If the input data is incorrect or inconsistent, the predictions may be incorrect.
- Limited to the data used for training: The predictions made by the solution are limited to the data used for training the machine learning model. If the data used for training is not representative of the data that the solution will be used for, the predictions may not be accurate.

8. APPLICATIONS:

The solution can be applied in a variety of industries, including energy, manufacturing, and agriculture, where predictions of energy consumption and demand are needed.

9. CONCLUSION:

The proposed solution provides a fast and accurate way to make predictions using machine learning algorithms. The solution is user-friendly, easy to implement, and does not require specialized knowledge. However, the quality of the predictions is dependent on the quality of the input data and the data used for training the machine learning model.

10. FUTURE SCOPE:

In the future, the solution can be enhanced to use more advanced machine learning algorithms and to incorporate additional input data to make more accurate predictions. The solution can also be expanded to make predictions in other areas, such as weather patterns or stock prices.

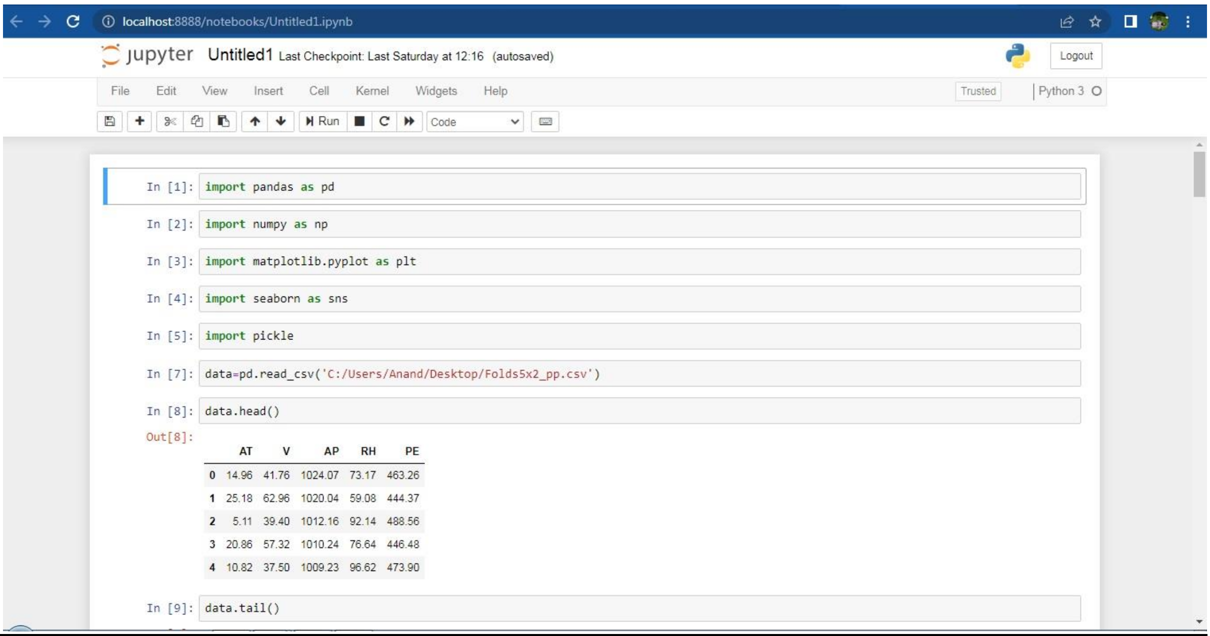
11. BIBLIOGRAPHY:

Flask framework documentation (https://flask.palletsprojects.com/en/2.1.x/)

Machine learning algorithms (https://en.wikipedia.org/wiki/Machine_learning)

Pickle library documentation (https://docs.python.org/3/library/pickle.html)

12. APPENDIX:



localhost:8888/notebooks/Untitled1.ipynb

jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [9]:
```
data.tail()
```
Out[9]:

	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

In [10]:
```
data.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9568 entries, 0 to 9567
Data columns (total 5 columns):
AT    9568 non-null float64
V     9568 non-null float64
AP    9568 non-null float64
RH    9568 non-null float64
PE    9568 non-null float64
dtypes: float64(5)
memory usage: 373.8 KB
```

In [11]:
```
data.describe()
```

jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Out[11]:

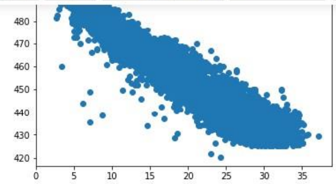
	AT	V	AP	RH	PE
count	9568.000000	9568.000000	9568.000000	9568.000000	9568.000000
mean	19.651231	54.305804	1013.259078	73.308978	454.365009
std	7.452473	12.707893	5.938784	14.600269	17.066995
min	1.810000	25.360000	992.890000	25.560000	420.260000
25%	13.510000	41.740000	1009.100000	63.327500	439.750000
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

In [12]:
```
data.isnull().sum()
```
Out[12]:
```
AT    0
V     0
AP    0
RH    0
PE    0
dtype: int64
```

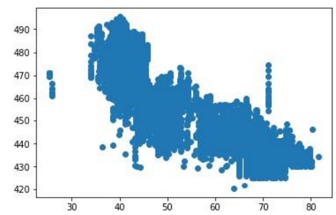
In [13]:
```
plt.scatter(data['AT'],data['PE'])
```
Out[13]: `<matplotlib.collections.PathCollection at 0x5f024a8>`

jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3



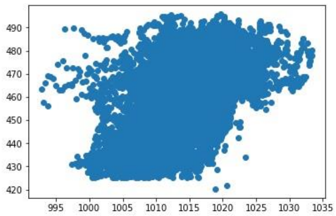
In [14]:
```
plt.scatter(data['V'],data['PE'])
```
Out[14]: `<matplotlib.collections.PathCollection at 0x5fa1a58>`



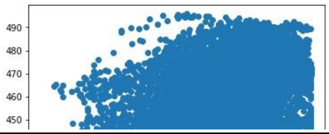
jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

```
In [15]: plt.scatter(data['AP'],data['PE'])
Out[15]: <matplotlib.collections.PathCollection at 0xbabb5f8>
```



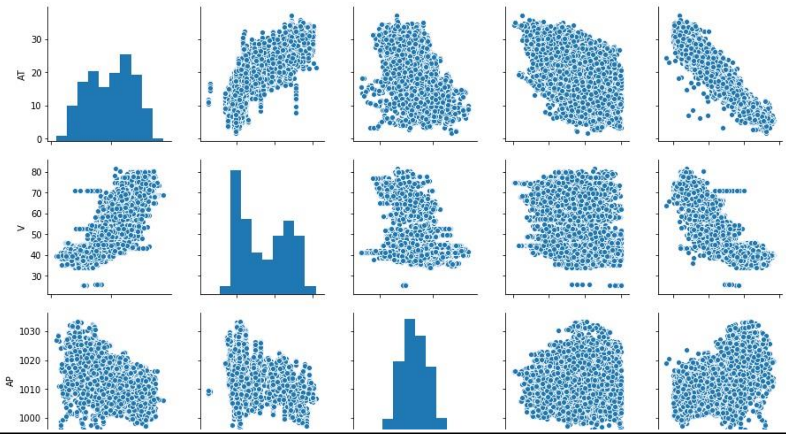
```
In [16]: plt.scatter(data['RH'],data['PE'])
Out[16]: <matplotlib.collections.PathCollection at 0xbb1eb38>
```



jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved)

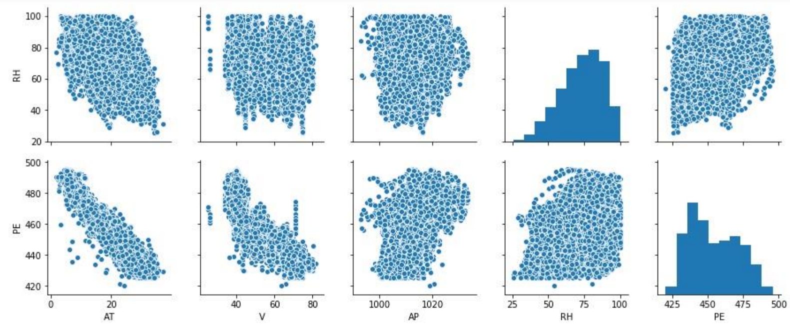
File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

```
In [17]: sns.pairplot(data,diag_kind = 'hist')
Out[17]: <seaborn.axisgrid.PairGrid at 0xa79de80>
```



jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3



```
In [18]: x=data.drop(['PE'],axis=1)
         y=data['PE']

In [19]: from sklearn.model_selection import train_test_split
         xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2, random_state = 0)

In [20]: xtrain.shape
```

jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [20]: xtrain.shape
Out[20]: (7654, 4)

In [21]: xtest.shape
Out[21]: (1914, 4)

In [22]: from sklearn.linear_model import LinearRegression
         from sklearn.tree import DecisionTreeRegressor

In [23]: from sklearn.ensemble import RandomForestRegressor

In [24]: LRmodel=LinearRegression()
         DTmodel=DecisionTreeRegressor()
         RFmodel=RandomForestRegressor()

In [25]: from sklearn.linear_model import LinearRegression
         LRmodel = LinearRegression()

In [26]: LRmodel.fit(xtrain, ytrain)
Out[26]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)

In [27]: LinearRegression()
Out[27]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)
```

jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
Out[27]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)

In [28]: LRpred=LRmodel.predict(xtest)
         from sklearn.metrics import r2_score
         LRscore=r2_score(ytest,LRpred)
         LRscore
Out[28]: 0.9325315554761302

In [29]: from sklearn.metrics import r2_score
         LRscore=r2_score(ytest, LRpred)
         LRscore
Out[29]: 0.9325315554761302

In [32]: from sklearn.tree import DecisionTreeRegressor
         DTRmodel=DecisionTreeRegressor()
         DTRmodel.fit(xtrain, ytrain)
         DecisionTreeRegressor()
         DTRpred=DTRmodel.predict(xtest)
         DTRscore=r2_score(ytest, DTRpred)
         DTRscore
Out[32]: 0.9204364213903032

In [33]: DTRscore=r2_score(ytest, DTRpred)
         DTRscore
Out[33]: 0.9204364213903032
```

jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [35]: from sklearn.ensemble import RandomForestRegressor
         RFmodel=RandomForestRegressor()
         RFmodel.fit(xtrain, ytrain)
         RandomForestRegressor()
         RFpred=RFmodel.predict(xtest)
         RFscore=r2_score(ytest, RFpred)
         RFscore

         C:\Users\Anand\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value of n_estimators wil
         l change from 10 in version 0.20 to 100 in 0.22.
           "10 in version 0.20 to 100 in 0.22.", FutureWarning)
Out[35]: 0.9592872800148419

In [36]: RFscore=r2_score(ytest, RFpred)
         RFscore
Out[36]: 0.9592872800148419

In [37]: from sklearn.metrics import r2_score
         LRscore=r2_score(ytest, LRpred)
         LRscore
Out[37]: 0.9325315554761302

In [38]: DTRscore=r2_score(ytest, DTRpred)
         DTRscore
Out[38]: 0.9204364213903032

In [41]: RFscore=r2_score(ytest, RFpred)
```

Jupyter Untitled1 Last Checkpoint: Last Saturday at 12:16 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
In [36]: RFscore=r2_score(ytest, RFpred)
         RFscore

Out[36]: 0.9592872800148419

In [37]: from sklearn.metrics import r2_score
         LRscore=r2_score(ytest, LRpred)
         LRscore

Out[37]: 0.9325315554761302

In [38]: DTRscore=r2_score(ytest, DTRpred)
         DTRscore

Out[38]: 0.9204364213903032

In [41]: RFscore=r2_score(ytest, RFpred)
         RFscore

Out[41]: 0.9592872800148419

In [42]: pickle.dump(RFmodel, open('CCPP.pkl','wb'))

In [ ]:
```