UPES

UNIVERSITY OF TOMORROW

Elements of AIML

ASSIGNMENT 1

Name: Somya Vats

Batch: 11

SAP: 500119012

Roll no: R2142230349

PREDICTING SOLAR PLANT YIELD USING MACHINE LEARNING ON WEATHER AND POWER DATA

- *GOOGLE DRIVE FOLDER*
- *GITHUB LINK*

Introduction

As climate change concerns continue to grow, global reliance on renewable sources has been increasingly sought out in order to not only follow a greener path, but to specifically address Sustainable Development Goals – SDG 7: Affordable and Clean Energy, and SDG 13: Climate Action. Solar energy provides clean energy, but it remains a hard area to maximise due to the uncontrollable factors of nature, which ultimately leads to variability in solar generation. All these factors make solar power forecasting an indispensable task when it comes to efficient energy management systems, plannings, and resource utilizations.

In this project, we focus on estimating how much energy a solar plant can produce by looking at past generation of the plant and weather conditions. Through the use of python ML models, the objective is to build a model that manages to forecast solar generation for a given weather scenario. Such models are a useful piece of machinery for solar plant operators as they can enhance the decision making by improving resource distribution and assisting in sustainable energy consumption as well.

This task is dependent on using Machine Learning given its sophistication in recognizing intricate structures within extensive amounts of data, self adjusting in real time, and providing forecast abilities which would be impossible otherwise. This project shows the other side of ML: how it changes the game for the renewable business.

Problem Statement

The production of solar energy is subject to the whims of sunlight, temperature, and cloud cover, making it an unreliable source that is difficult for grid operators to effectively budget for. This variability can threaten grid stability, make energy storage planning difficult, and create energy distribution inefficiencies, especially with the growing worldwide need for renewable energy.

The International Renewable Energy Agency (IRENA) says that solar energy will need to increase dramatically in order to meet climate action goals, possibly increasing by a factor of ten by the year 2050. In order to satisfy these requirements and to increase reliability, precise solar yield forecasts are necessary. This project will use machine learning and historical weather and power generation data to predict how much a solar plant will yield, so that the operators can manage the flow of energy, maximize the stability of the grid, and reach energy supply goals.

Solution Overview

Based on the examination of past weather statistics and related energy generation data, this project proposes a machine learning technique for estimating the total energy output for a solar plant. To achieve this goal the model inputs four environmental variables; sunlight intensity, cloud cover, temperature and humidity. This enables solar power plant managers to optimize their choices for energy storage, integration into the grid and investment level for deployment resources.

The time-series data contains information about the solar power output and the changes in the atmosphere including different weather conditions such as day and night and the seasons is what the model is trained upon. Then this model is applied to predict energy generation during the next hours while attempting to balance energy demand and supply in the grid. This makes it possible for the operators to appropriately prepare for high production periods and low production periods.

The flexibility of the model means that it can be transferred across various locations and applied to other similar solar plants with appropriate geographical data. Also, since the system assimilates additional information, its precision and flexibility enhance over time, making it an effective tool for the renewable energy sector to meet optimized productivity and efficiency.

Tech Stack

This application was developed with the help of HTML/CSS and Flask as a framework for process-oriented model training and interaction with users, along with the combination of several libraries of the Python programming language.

Python Libraries:

I. Pandas and NumPy were used for data handling and storage as well as for statistical data analysis, that required dealing with big amounts of data.
II. Scikit-learn was utilized for performing most of the machine learning algorithms, model training and hyperparameters tuning.
III. Matplotlib and seaborn libraries aimed to illustrate data and models results in graphical representations.

Flask (Backend): As the backend framework application, Flask offered the simplest integration tool to the developers as minimalistic API routes existed in order to make predictions and query data.

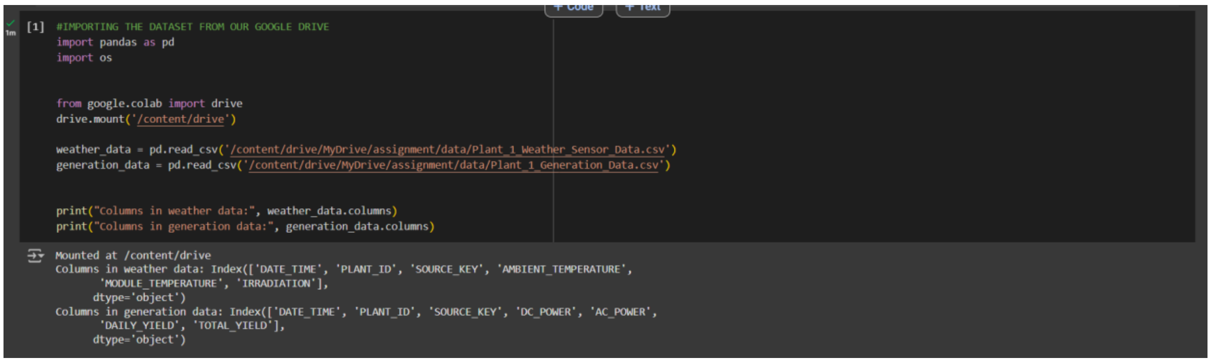
HTML/CSS (Frontend): Used to construct model visualizations and model forecast outputs allowing users to perform prediction tasks and other engaged procedures actively and simply and look at descriptive parts of the data such as images.

CODE DESCRIPTION

- *GOOGLE DRIVE FOLDER*
- *GITHUB LINK*

STEP 1 DATA ACQUIRING

1. *Dataset has been taken from kaggle and uploaded into google drive as a ".csv" file*
2. *Drive is imported and so is the data.*
3. *Then, just to check the success of these steps, few content is printed.*



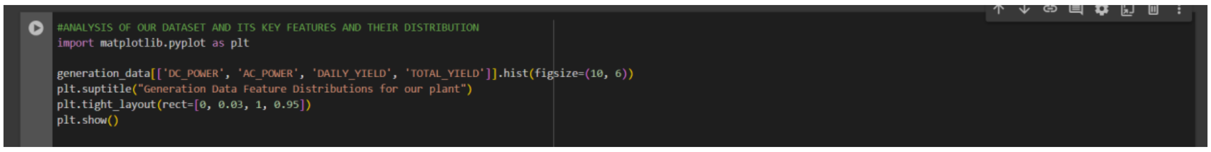
```
[1] #IMPORTING THE DATASET FROM OUR GOOGLE DRIVE
    import pandas as pd
    import os


    from google.colab import drive
    drive.mount('/content/drive')

    weather_data = pd.read_csv('/content/drive/MyDrive/assignment/data/Plant_1_Weather_Sensor_Data.csv')
    generation_data = pd.read_csv('/content/drive/MyDrive/assignment/data/Plant_1_Generation_Data.csv')


    print("Columns in weather data:", weather_data.columns)
    print("Columns in generation data:", generation_data.columns)
```

```
Mounted at /content/drive
Columns in weather data: Index(['DATE_TIME', 'PLANT_ID', 'SOURCE_KEY', 'AMBIENT_TEMPERATURE',
       'MODULE_TEMPERATURE', 'IRRADIATION'],
      dtype='object')
Columns in generation data: Index(['DATE_TIME', 'PLANT_ID', 'SOURCE_KEY', 'DC_POWER', 'AC_POWER',
       'DAILY_YIELD', 'TOTAL_YIELD'],
      dtype='object')
```

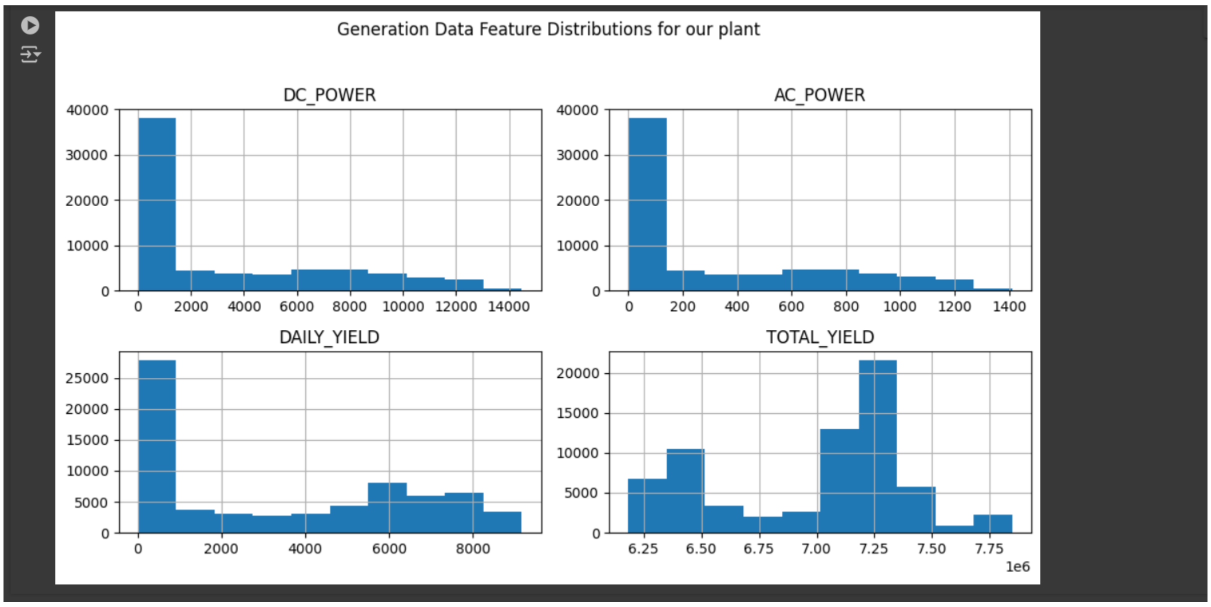
STEP 2 VISUALIZATION OF CURRENT DATA'S TREND AND FEATURES

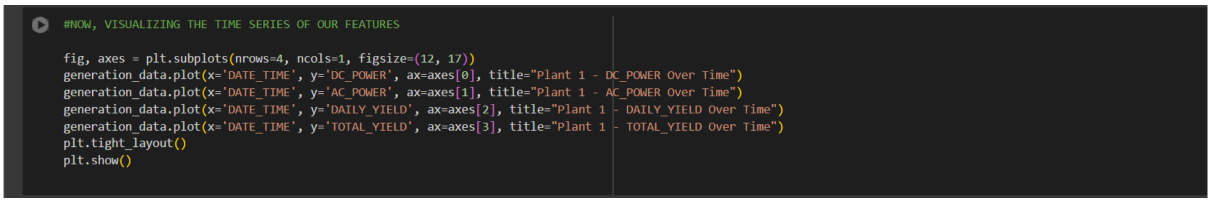
1. Checking the distribution of the features of the data

2. Then, checking the time series trends, as later, we merge both our dataset based off the "Date_time" column

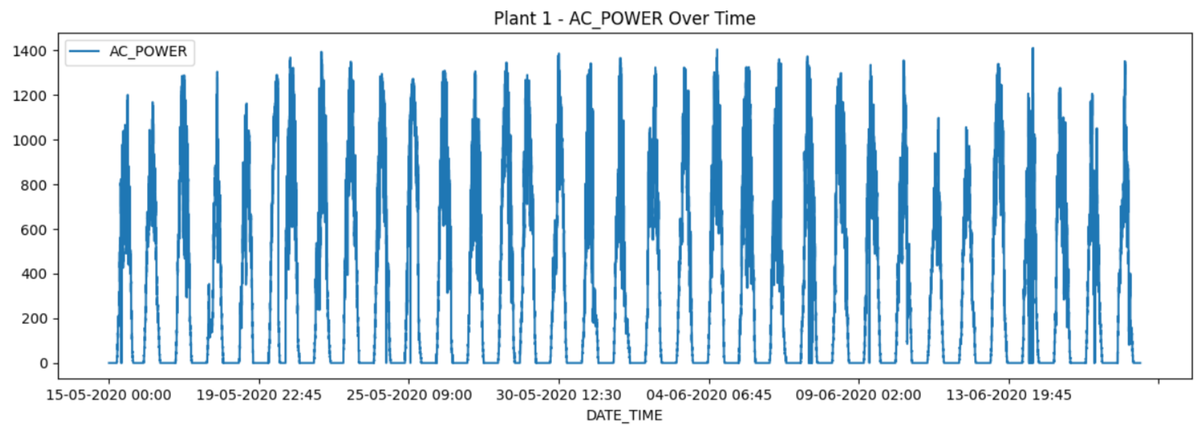


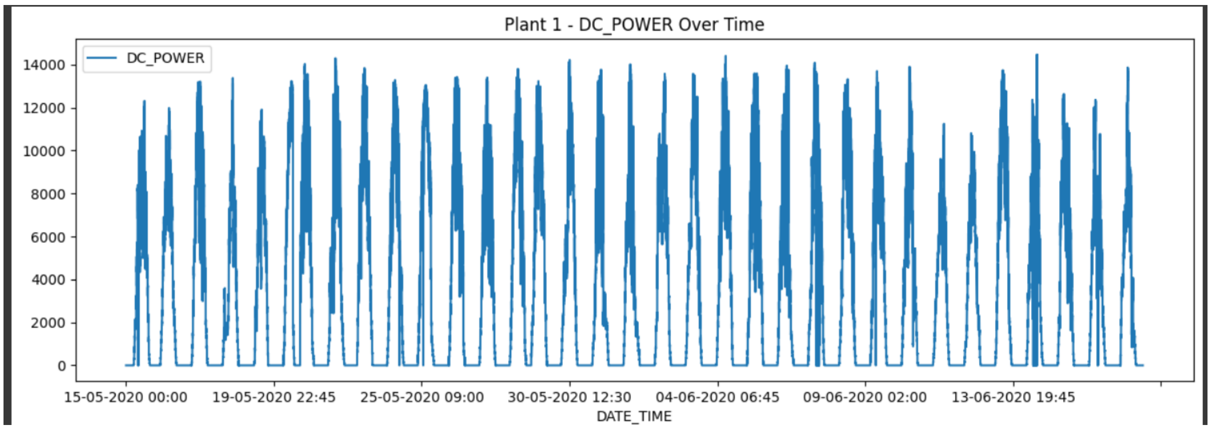
```
#ANALYSIS OF OUR DATASET AND ITS KEY FEATURES AND THEIR DISTRIBUTION
import matplotlib.pyplot as plt

generation_data[['DC_POWER', 'AC_POWER', 'DAILY_YIELD', 'TOTAL_YIELD']].hist(figsize=(10, 6))
plt.suptitle("Generation Data Feature Distributions for our plant")
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

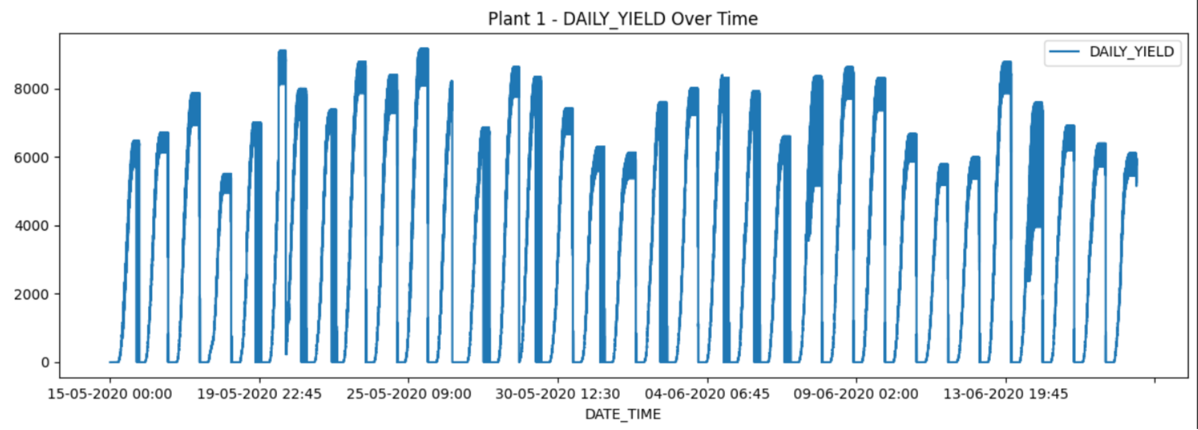


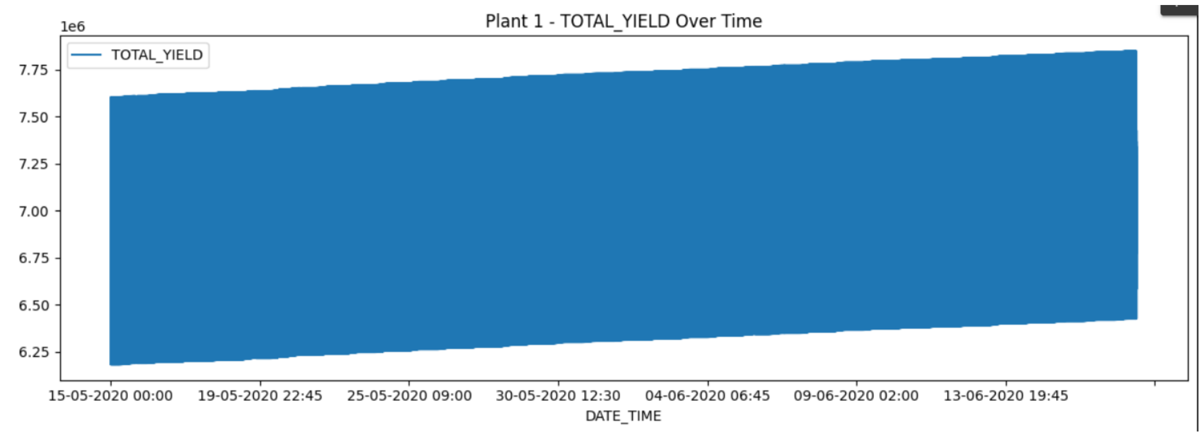


```
#NOW, VISUALIZING THE TIME SERIES OF OUR FEATURES

fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(12, 17))
generation_data.plot(x='DATE_TIME', y='DC_POWER', ax=axes[0], title="Plant 1 - DC_POWER Over Time")
generation_data.plot(x='DATE_TIME', y='AC_POWER', ax=axes[1], title="Plant 1 - AC_POWER Over Time")
generation_data.plot(x='DATE_TIME', y='DAILY_YIELD', ax=axes[2], title="Plant 1 - DAILY_YIELD Over Time")
generation_data.plot(x='DATE_TIME', y='TOTAL_YIELD', ax=axes[3], title="Plant 1 - TOTAL_YIELD Over Time")
plt.tight_layout()
plt.show()
```


Plant 1 - AC_POWER Over Time
AC_POWER
1400
1200
1000
800
600
400
200
0
15-05-2020 00:00
19-05-2020 22:45
25-05-2020 09:00
30-05-2020 12:30
04-06-2020 06:45
09-06-2020 02:00
13-06-2020 19:45
DATE_TIME


Plant 1 - DC_POWER Over Time
DC_POWER
14000
12000
10000
8000
6000
4000
2000
0
15-05-2020 00:00
19-05-2020 22:45
25-05-2020 09:00
30-05-2020 12:30
04-06-2020 06:45
09-06-2020 02:00
13-06-2020 19:45
DATE_TIME


Plant 1 - DAILY_YIELD Over Time
DAILY_YIELD
8000
6000
4000
2000
0
15-05-2020 00:00
19-05-2020 22:45
25-05-2020 09:00
30-05-2020 12:30
04-06-2020 06:45
09-06-2020 02:00
13-06-2020 19:45
DATE_TIME


Plant 1 - TOTAL_YIELD Over Time
1e6
TOTAL_YIELD
7.75
7.50
7.25
7.00
6.75
6.50
6.25
15-05-2020 00:00
19-05-2020 22:45
25-05-2020 09:00
30-05-2020 12:30
04-06-2020 06:45
09-06-2020 02:00
13-06-2020 19:45
DATE_TIME

STEP 3. DATA PROCESSING

1. We change the format of the "DATE_TIME" column into the datetime data format in python

2. Then, we check the data types and missing values for both the datasets

3. Since generation data is huge, we condense it

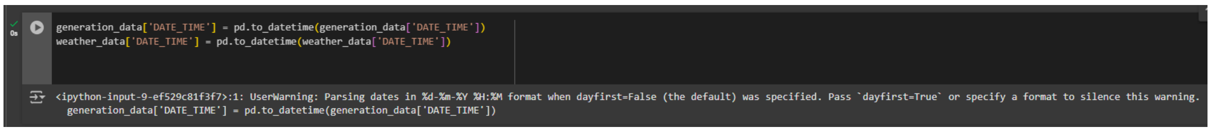
4. Then, on the basis of "DATE_TIME" and "PLANT_ID", merge both the datasets as the prediction will be made on basis of combined features

4. We condense "merged_data" into a sub-dataset as the initial one was very large size

5. We distribute the train and validation splits over the data

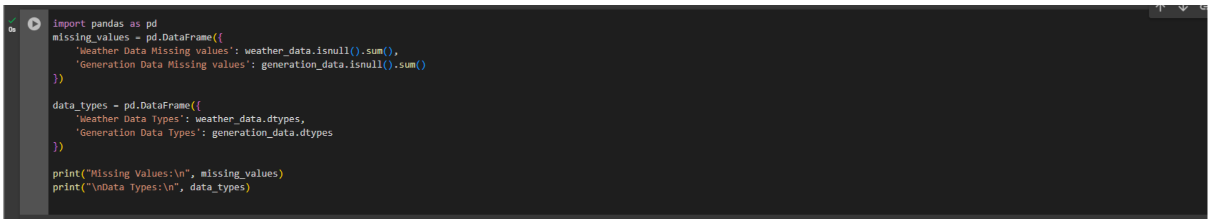
6. We check if we require SMOTE application for our data

7. We further check the class distribution for imbalances, so to further check the applicabilty of SMOGN, etc

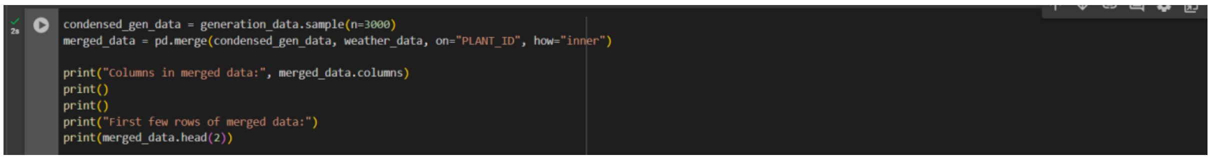


```
generation_data['DATE_TIME'] = pd.to_datetime(generation_data['DATE_TIME'])
weather_data['DATE_TIME'] = pd.to_datetime(weather_data['DATE_TIME'])
```

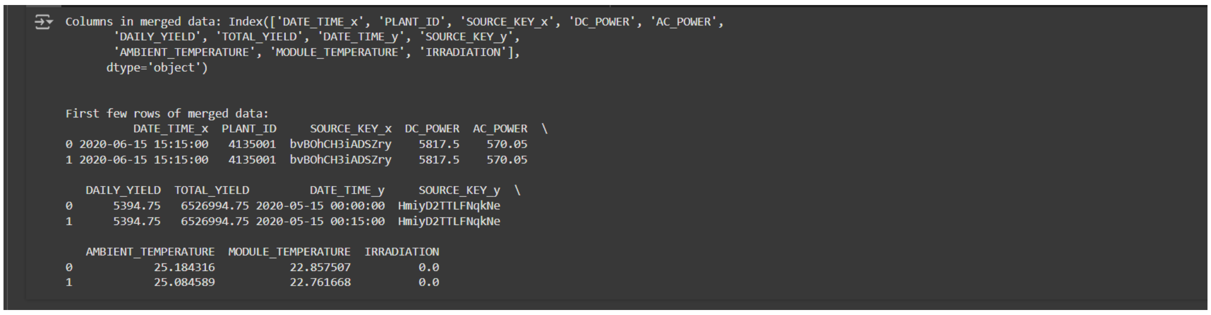
```
<ipython-input-9-ef529c81f3f7>:1: UserWarning: Parsing dates in %d-%m-%Y %H:%M format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence this warning.
  generation_data['DATE_TIME'] = pd.to_datetime(generation_data['DATE_TIME'])
```



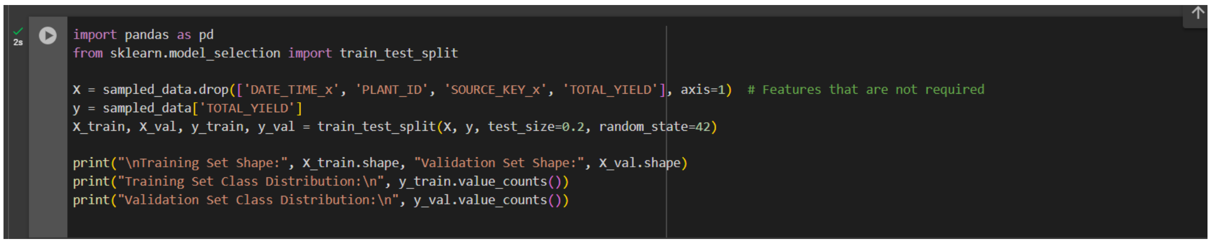
```
import pandas as pd
missing_values = pd.DataFrame({
    'Weather Data Missing values': weather_data.isnull().sum(),
    'Generation Data Missing values': generation_data.isnull().sum()
})

data_types = pd.DataFrame({
    'Weather Data Types': weather_data.dtypes,
    'Generation Data Types': generation_data.dtypes
})

print("Missing Values:\n", missing_values)
print("\nData Types:\n", data_types)
```



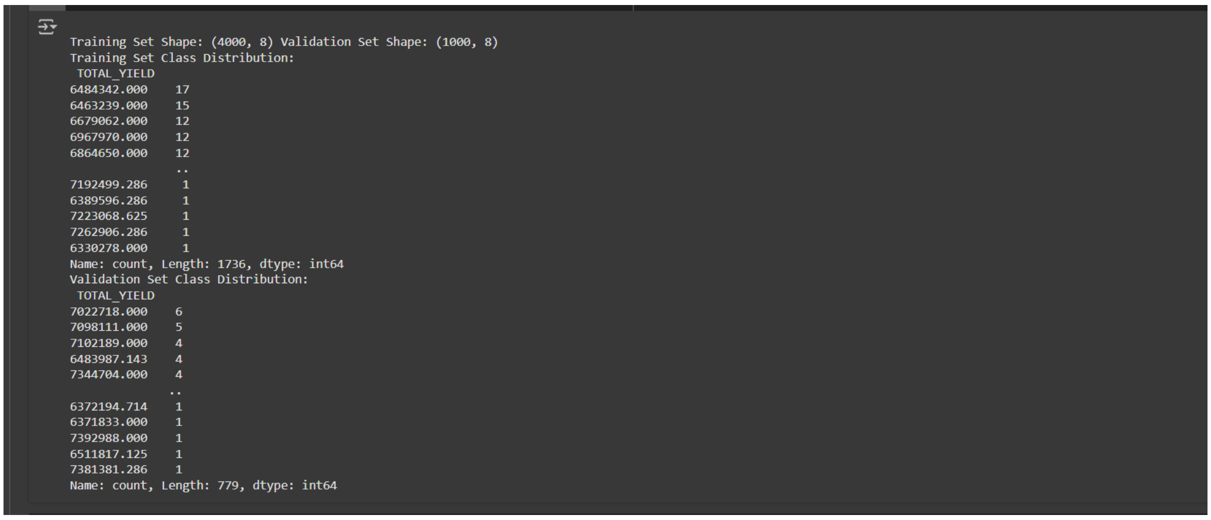
```
condensed_gen_data = generation_data.sample(n=3000)
merged_data = pd.merge(condensed_gen_data, weather_data, on="PLANT_ID", how="inner")

print("Columns in merged data:", merged_data.columns)
print()
print()
print("First few rows of merged data:")
print(merged_data.head(2))
```



```
Columns in merged data: Index(['DATE_TIME_x', 'PLANT_ID', 'SOURCE_KEY_x', 'DC_POWER', 'AC_POWER',
       'DAILY_YIELD', 'TOTAL_YIELD', 'DATE_TIME_y', 'SOURCE_KEY_y',
       'AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE', 'IRRADIATION'],
      dtype='object')


First few rows of merged data:
          DATE_TIME_x  PLANT_ID     SOURCE_KEY_x  DC_POWER  AC_POWER  \
0 2020-06-15 15:15:00   4135001  bvBOhCH3iADSZry    5817.5    570.05
1 2020-06-15 15:15:00   4135001  bvBOhCH3iADSZry    5817.5    570.05

   DAILY_YIELD  TOTAL_YIELD         DATE_TIME_y     SOURCE_KEY_y  \
0      5394.75   6526994.75 2020-05-15 00:00:00  HmiyD2TTLFNqkNe
1      5394.75   6526994.75 2020-05-15 00:15:00  HmiyD2TTLFNqkNe

   AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  IRRADIATION
0            25.184316           22.857507          0.0
1            25.084589           22.761668          0.0
```

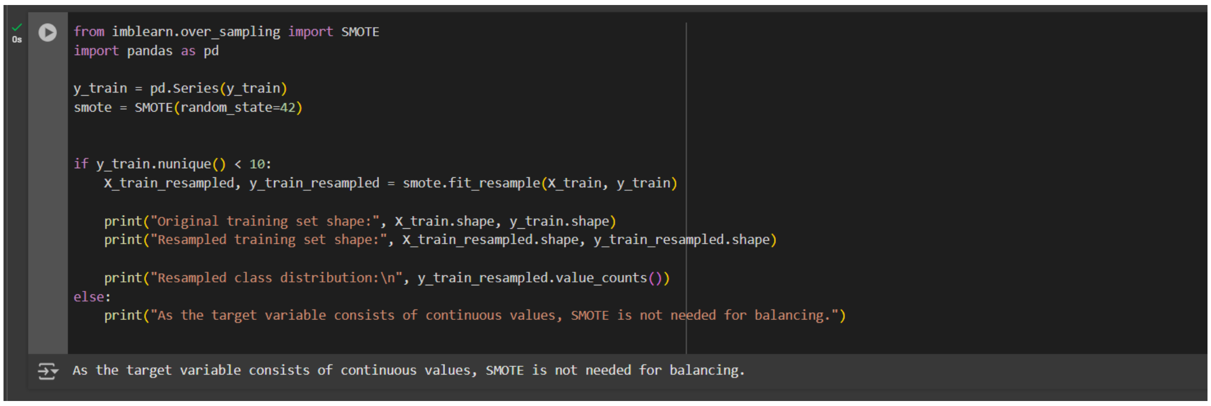


```python
import pandas as pd
from sklearn.model_selection import train_test_split

X = sampled_data.drop(['DATE_TIME_x', 'PLANT_ID', 'SOURCE_KEY_x', 'TOTAL_YIELD'], axis=1)  # Features that are not required
y = sampled_data['TOTAL_YIELD']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

print("\nTraining Set Shape:", X_train.shape, "Validation Set Shape:", X_val.shape)
print("Training Set Class Distribution:\n", y_train.value_counts())
print("Validation Set Class Distribution:\n", y_val.value_counts())
```



```
Training Set Shape: (4000, 8) Validation Set Shape: (1000, 8)
Training Set Class Distribution:
 TOTAL_YIELD
6484342.000    17
6463239.000    15
6679062.000    12
6967970.000    12
6864650.000    12
               ..
7192499.286     1
6389596.286     1
7223068.625     1
7262906.286     1
6330278.000     1
Name: count, Length: 1736, dtype: int64
Validation Set Class Distribution:
 TOTAL_YIELD
7022718.000    6
7098111.000    5
7102189.000    4
6483987.143    4
7344704.000    4
              ..
6372194.714    1
6371833.000    1
7392988.000    1
6511817.125    1
7381381.286    1
Name: count, Length: 779, dtype: int64
```

***SMOTE** is not applied to the continuous data*

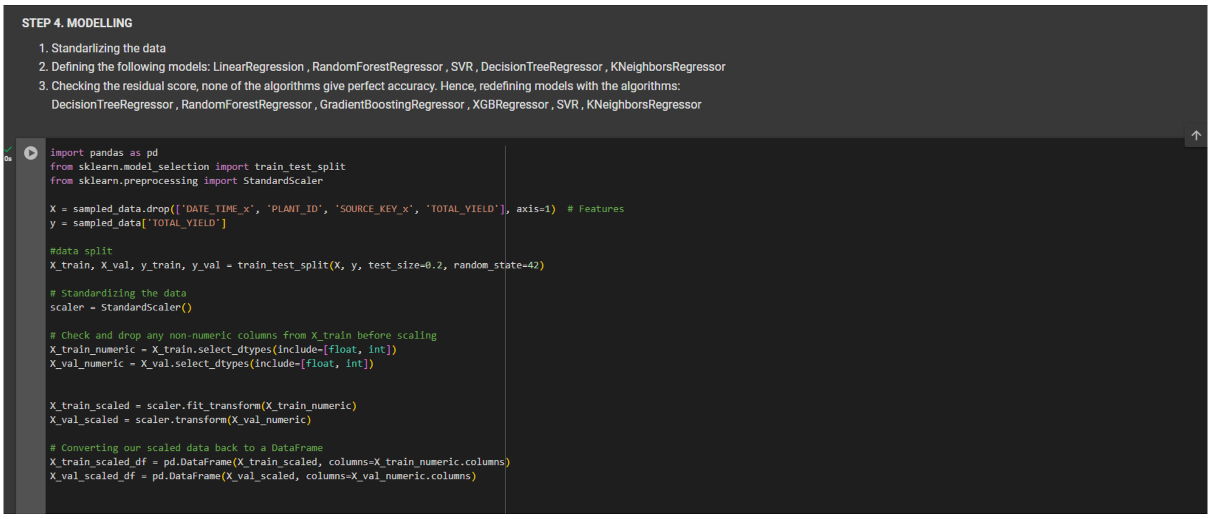


```python
from imblearn.over_sampling import SMOTE
import pandas as pd

y_train = pd.Series(y_train)
smote = SMOTE(random_state=42)


if y_train.nunique() < 10:
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

    print("Original training set shape:", X_train.shape, y_train.shape)
    print("Resampled training set shape:", X_train_resampled.shape, y_train_resampled.shape)

    print("Resampled class distribution:\n", y_train_resampled.value_counts())
else:
    print("As the target variable consists of continuous values, SMOTE is not needed for balancing.")
```

```
As the target variable consists of continuous values, SMOTE is not needed for balancing.
```

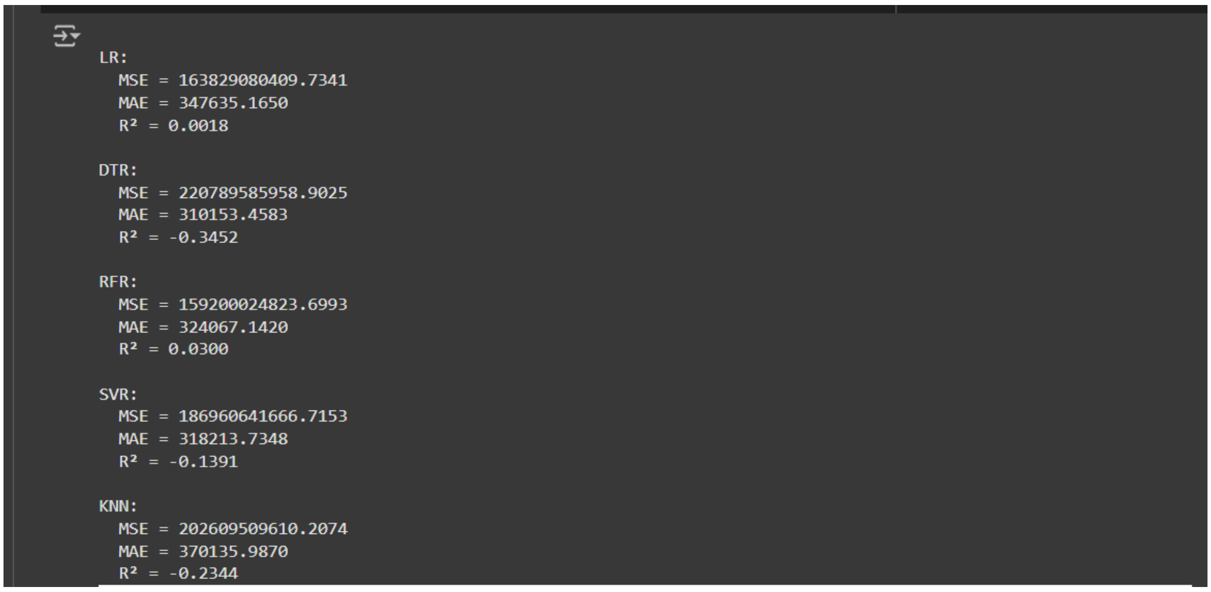
STEP 4. MODELLING

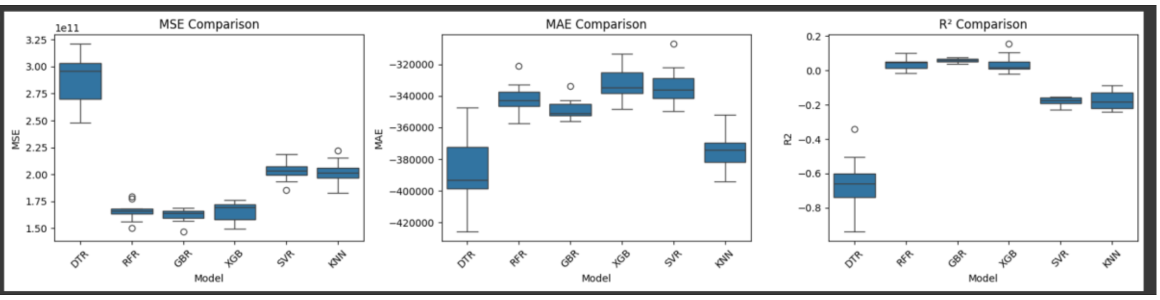
1. *Standarlizing the data*
2. *Defining the following models:*
3. *Checking the residual score*

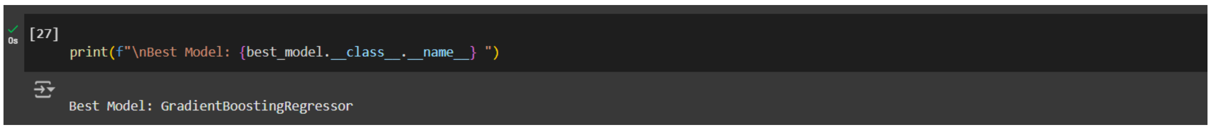

STEP 4. MODELLING

1. Standarlizing the data
2. Defining the following models: LinearRegression , RandomForestRegressor , SVR , DecisionTreeRegressor , KNeighborsRegressor
3. Checking the residual score, none of the algorithms give perfect accuracy. Hence, redefining models with the algorithms: DecisionTreeRegressor , RandomForestRegressor , GradientBoostingRegressor , XGBRegressor , SVR , KNeighborsRegressor

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = sampled_data.drop(['DATE_TIME_x', 'PLANT_ID', 'SOURCE_KEY_x', 'TOTAL_YIELD'], axis=1)  # Features
y = sampled_data['TOTAL_YIELD']

#data split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the data
scaler = StandardScaler()

# Check and drop any non-numeric columns from X_train before scaling
X_train_numeric = X_train.select_dtypes(include=[float, int])
X_val_numeric = X_val.select_dtypes(include=[float, int])


X_train_scaled = scaler.fit_transform(X_train_numeric)
X_val_scaled = scaler.transform(X_val_numeric)

# Converting our scaled data back to a DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train_numeric.columns)
X_val_scaled_df = pd.DataFrame(X_val_scaled, columns=X_val_numeric.columns)
```


```
LR:
  MSE = 163829080409.7341
  MAE = 347635.1650
  R² = 0.0018

DTR:
  MSE = 220789585958.9025
  MAE = 310153.4583
  R² = -0.3452

RFR:
  MSE = 159200024823.6993
  MAE = 324067.1420
  R² = 0.0300

SVR:
  MSE = 186960641666.7153
  MAE = 318213.7348
  R² = -0.1391

KNN:
  MSE = 202609509610.2074
  MAE = 370135.9870
  R² = -0.2344
```

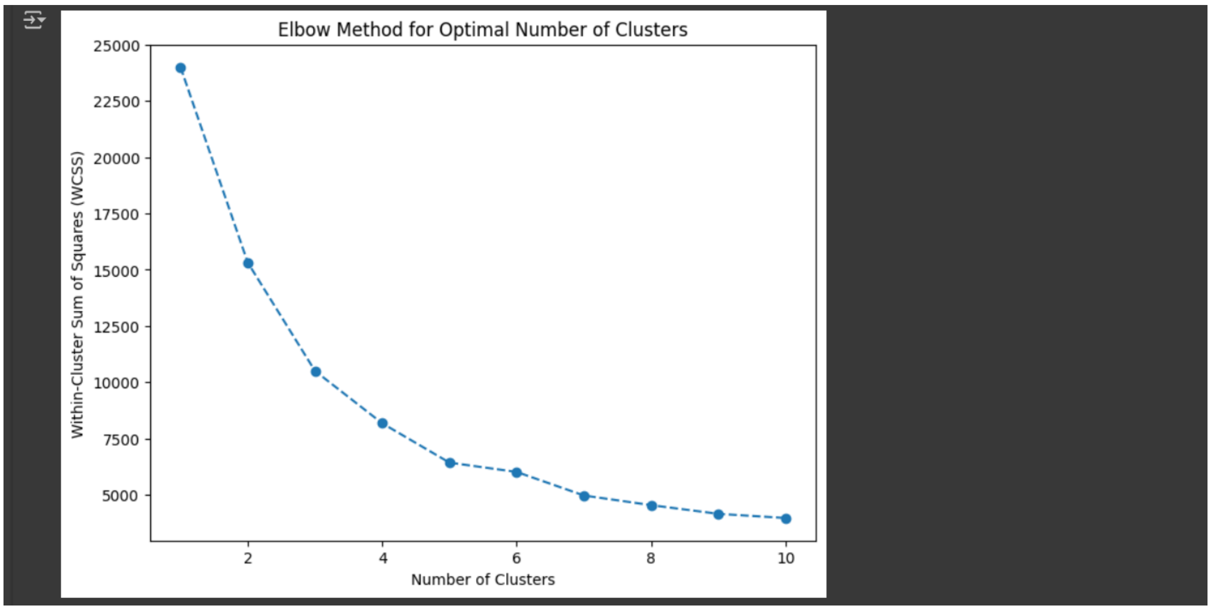


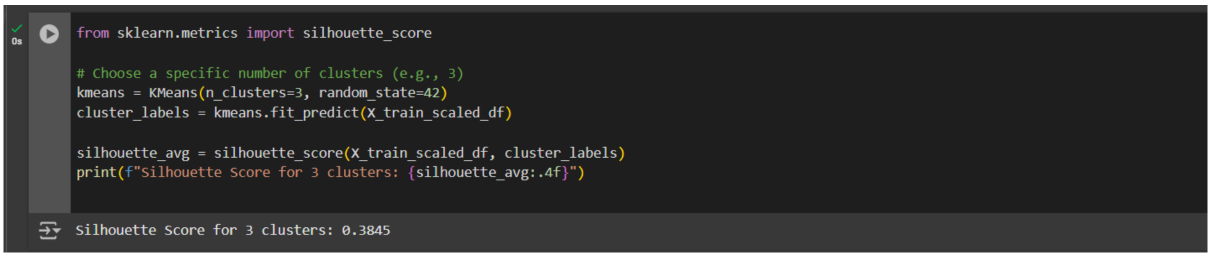

```
[27]
print(f"\nBest Model: {best_model.__class__.__name__} ")

Best Model: GradientBoostingRegressor
```

STEP 6. CLUSTER MEANS AND SILHOUTTE SCORE

1. *Optimal number of clusters*

2. *Silhoutte score*



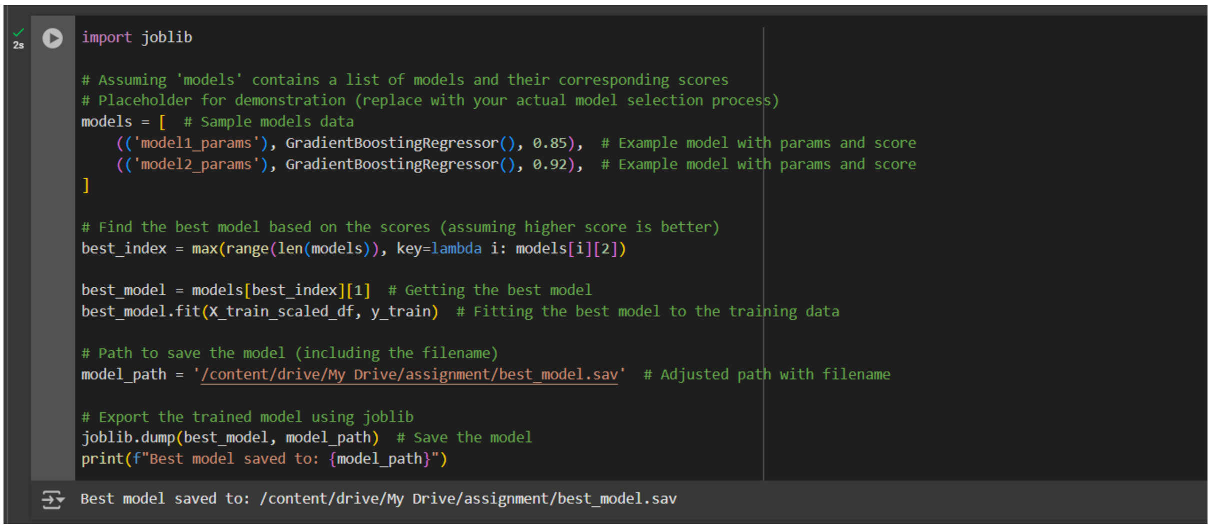


```
from sklearn.metrics import silhouette_score

# Choose a specific number of clusters (e.g., 3)
kmeans = KMeans(n_clusters=3, random_state=42)
cluster_labels = kmeans.fit_predict(X_train_scaled_df)

silhouette_avg = silhouette_score(X_train_scaled_df, cluster_labels)
print(f"Silhouette Score for 3 clusters: {silhouette_avg:.4f}")
```

```
Silhouette Score for 3 clusters: 0.3845
```

STEP 6. BEST MODEL FOR PREDICTION

1. *Checking the maximum and minimum range for our input*
2. *Submitting a test data (input) and predicting the total yield based off our best model*
3. *Saving our best model into the google drive folder*

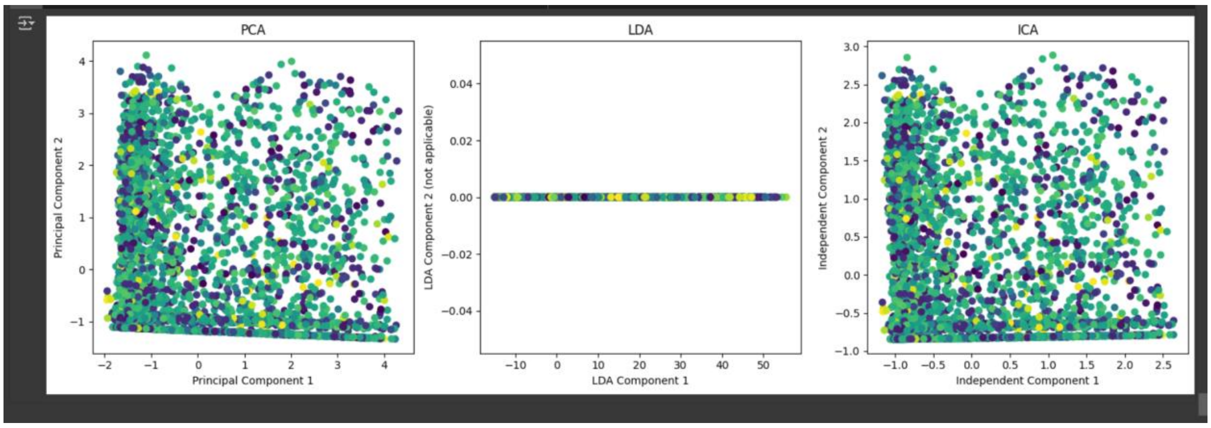


```
import joblib

# Assuming 'models' contains a list of models and their corresponding scores
# Placeholder for demonstration (replace with your actual model selection process)
models = [  # Sample models data
    (('model1_params'), GradientBoostingRegressor(), 0.85),  # Example model with params and score
    (('model2_params'), GradientBoostingRegressor(), 0.92),  # Example model with params and score
]

# Find the best model based on the scores (assuming higher score is better)
best_index = max(range(len(models)), key=lambda i: models[i][2])

best_model = models[best_index][1]  # Getting the best model
best_model.fit(X_train_scaled_df, y_train)  # Fitting the best model to the training data

# Path to save the model (including the filename)
model_path = '/content/drive/My Drive/assignment/best_model.sav'  # Adjusted path with filename

# Export the trained model using joblib
joblib.dump(best_model, model_path)  # Save the model
print(f"Best model saved to: {model_path}")
```

```
Best model saved to: /content/drive/My Drive/assignment/best_model.sav
```

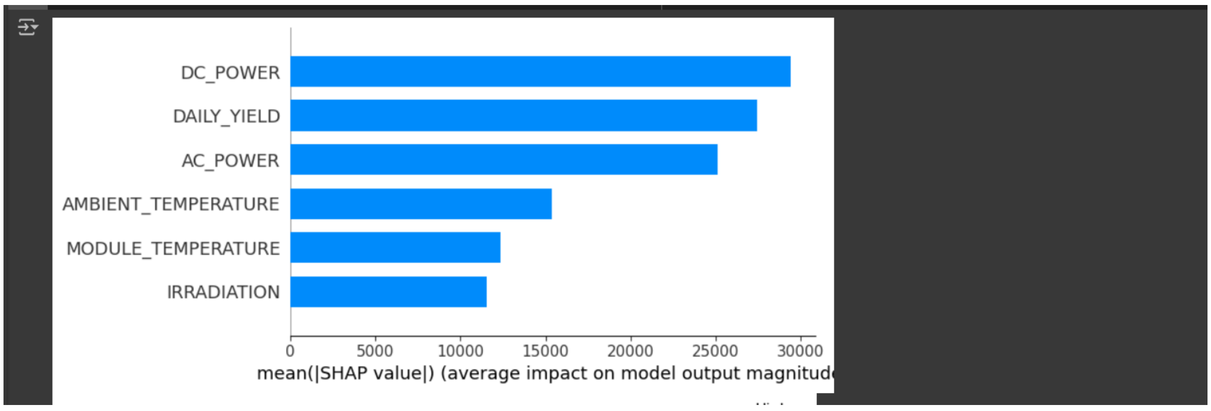
STEP 7. VISUALIZATION FOR DIMENSIONAL ANALYSIS

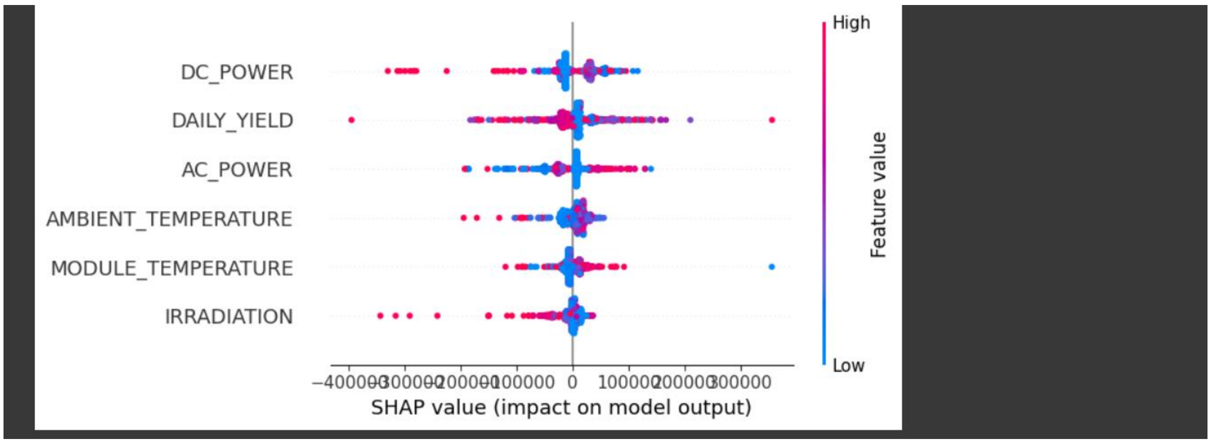
1. *Applying PDA algorithm*
2. *Applying LDA algorithm*
3. *Applying ICA algorithm*

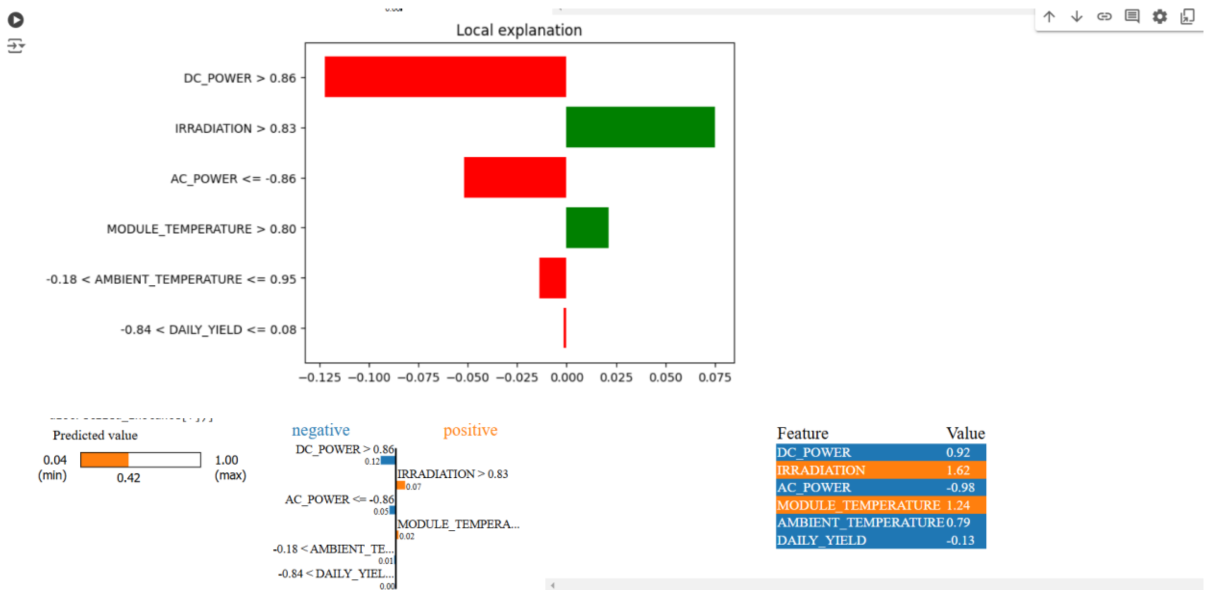


Step 8. Understanding our model with the help of Explainable AI models

1. *SHAP*
2. *LIME*







Feature	Value
DC_POWER	0.92
IRRADIATION	1.62
AC_POWER	-0.98
MODULE_TEMPERATURE	1.24
AMBIENT_TEMPERATURE	0.79
DAILY_YIELD	-0.13

Practical Applications

The model for predicting solar energy output has practical uses in the renewable energy industry as it supplies the operators, energy companies and policymakers involved in energy production with the necessary foresight. Such predictions ensure effective and efficient energy production. The main advantages are the following:

1. **Higher Efficiency:** When the operators are able to determine the amount of energy to be generated accurately, they can change the production plans in such a way as to optimize efficiency and reduce wastage of energy resources.
2. **Cost Saving:** Factors such as overproduction or underproduction have great operational cost implications due to the need to install energy storage systems or balance the power grid.
3. **Improved Grid Orchestration:** Accurate analyses also help install solar power plants scattered all over the grid as their supporting roles, enhancing grid stability and reducing fossil energy uses.
4. **Sustainability Orientation**: The constructs of the solution endorse energy companies to attain the sustainability agenda as they adopt the low carbon emission options which make solar energy competitive in the energy mix. Such a predictive solution also has a significant impact to other sectors that are not directly associated with renewable energy.

For instance, hospitals that mostly rely on renewable energy sources for emergency needs can be assured of reliability in supply due to the accurate estimation of yield. This model also has applicability such as in government agencies by which its use can determine the performance of solar PV capacity of a nation and thus provide guidance on renewable energy's policies and investment opportunities.

Key Takeaways and Challenges

Through this project, I gained insights into:

- **Technical Skills**: Enhanced understanding of data preprocessing, feature selection, and working with regression models to address real-world problems.
- **Conceptual Insights**: The project emphasized the connection between weather patterns and energy production, illustrating how ML models can leverage these insights.

Key challenges included:

1. **Data Quality**: Cleaning and preprocessing data to handle missing values and inconsistencies.
2. **Model Accuracy**: Initial predictions were suboptimal, requiring model experimentation and hyperparameter tuning to improve accuracy.

Acknowledgements

I am grateful for my mentor's guidance throughout this project, helping me understand core concepts in machine learning and solar energy forecasting. I also appreciate my peers for reviewing my work and providing valuable feedback.

References

1. Anikannal, "Solar Power Generation Data," Kaggle, Kaggle link.
2. J. Brownlee, "Machine Learning Mastery," ML Mastery
3. Scikit-learn Documentation, Scikit-learn link
4. VanderPlas, "Data Science Handbook," O'Reilly Media.
5. Springer, "Renewable Energy Forecasting," Springer Publications, 2016.

- *GOOGLE DRIVE FOLDER*
- *GITHUB LINK*

Name: Somya Vats

Batch: 11

SAP: 500119012

Roll no: R2142230349