**Global Power Plant Database**

**Project Overview:**

The Global Power Plant Database **is** a comprehensive, open-source database of power plants around the world**.** It centralizes power plant data to make it easier to navigate, compare **and** draw insights **for** one’s own analysis**.** The database covers approximately 14,000 power plants **from** 3 countries (USA, AUS, INDIA) **and** includes thermal plants**.**

This article will cover all key steps, from defining the problem to model building, evaluation, and hyperparameter tuning.

Aim to make two predictions **for** Primary fuel **and** capacity\_mw**.**

This article contains the topics are listed below:

1.  Problem Definition  
2.  Data Analysis  
3.  EDA Concluding Remarks  
4.  Pre-processing Pipeline  
5.  Building Machine Learning Models  
6.  Conclusion

**1. Problem Definition**

The **Global Power Plant Database** provides a comprehensive dataset containing detailed information on power plants across different countries. This project focuses on building machine learning models to make two key predictions based on this dataset:

1. **Classification of Primary Fuel Type** used by the power plant.
2. **Regression to predict the power capacity** (in megawatts) of the power plants.

The project aims to automate these predictions, leveraging the data available to provide insights into the types of power plants and their capacities. The machine learning models built in this project include a **Random Forest Classifier** for the primary fuel type prediction and a **Linear Regression model** for capacity prediction.

The Global Power Plant Database contains information about power plants worldwide, including their geographic location, primary fuel type, capacity, ownership, and more. However, some plants may have missing or incomplete data, especially concerning their primary fuel type and capacity. Therefore, the primary goals of this project are:

1. **Classification Task**: Predict the primary fuel type of a power plant.
2. **Regression Task**: Predict the power capacity (in megawatts) of a plant.

Accurately predicting the fuel type and capacity can be helpful in various scenarios, such as energy planning, analysis of the global energy market, and understanding regional energy trends.

Python program to complete the entire project, step-by-step:

1. **Importing Libraries**
2. **Loading Data**
3. **EDA (Exploratory Data Analysis)**
4. **Data Preprocessing**
5. **Feature Engineering**
6. **Model Building & Testing**
7. **Hyperparameter Tuning**
8. **Model Selection**
9. **Saving the Best Model**

**Process steps simple outline:**

Load Dataset: The dataset **is** loaded directly **from** a URL**.**

Exploratory Data Analysis (EDA): Basic insights are obtained, missing values are visualized, **and** correlations are explored**.**

Data Pre-processing:

Missing values are filled**.**

Categorical variables are encoded**.**

Feature Engineering:

Irrelevant columns are dropped**.**

Highly correlated features are removed**.**

Splitting Data:

The dataset **is** split **for** both classification **and** regression tasks**.**

Model Building:

Classification: RandomForestClassifier **is** used to predict primary fuel type**.**

Regression: LinearRegression **is** used to predict power capacity**.**

Model Evaluation:

Classification metrics include accuracy **and** classification report**.**

Regression metrics include mean squared error **and** R**-**squared**.**

Hyperparameter Tuning:

GridSearchCV **is** used to find the best parameters **for** both classification **and** regression models**.**

Model Saving:

The best models are saved using pickle **for** future use**.**

**2. Data Analysis**

**Data Overview**

The dataset used in this project contains 908 entries from 3 countries (USA, Australia, India) and features various types of power plants, both thermal and renewable. The data includes several columns, such as:

* **country**: 3-character country code.
* **primary\_fuel**: The primary energy source (e.g., coal, gas, nuclear, solar, etc.).
* **capacity\_mw**: The generating capacity in megawatts.
* **latitude, longitude**: Geolocation of the power plants.
* **commissioning\_year**: The year the plant began operating.
* **generation\_gwh\_2013** to generation\_gwh\_2019: Annual power generation data (in gigawatt-hours).

**Missing Values**

The dataset has missing values across various columns. For example, latitude and longitude have 46 missing values, while columns like other\_fuel1 and commissioning\_year also contain missing data. Given the incomplete nature of the data, proper handling of these missing values was a crucial step in data preprocessing.

**Data Distribution**

We examined the distributions of our target variables, which are **primary fuel** and **capacity\_mw**:

1. **Primary Fuel**: This categorical feature has multiple classes, such as gas, hydro, wind, solar, coal, and others. The distribution of fuel types across the dataset can be seen in the plot below:

python

Copy code

sns.countplot(x='primary\_fuel', data=data)

plt.show()

1. **Capacity (in MW)**: The capacity of power plants ranges widely, from very small renewable plants to massive thermal power plants generating thousands of megawatts:

python

Copy code

sns.histplot(data['capacity\_mw'], kde=True)

plt.show()

**Key attributes of the database**

The database includes the following indicators:

* `country` (text): 3 character country code corresponding to the ISO 3166-1 alpha-3 specification [5]
* `country\_long` (text): longer form of the country designation
* `name` (text): name or title of the power plant, generally in Romanized form
* `gppd\_idnr` (text): 10 or 12 character identifier for the power plant
* `capacity\_mw` (number): electrical generating capacity in megawatts
* `latitude` (number): geolocation in decimal degrees; WGS84 (EPSG:4326)
* `longitude` (number): geolocation in decimal degrees; WGS84 (EPSG:4326)
* `primary\_fuel` (text): energy source used in primary electricity generation or export
* `other\_fuel1` (text): energy source used in electricity generation or export
* `other\_fuel2` (text): energy source used in electricity generation or export
* `other\_fuel3` (text): energy source used in electricity generation or export
* `commissioning\_year` (number): year of plant operation, weighted by unit-capacity when data is available
* `owner` (text): majority shareholder of the power plant, generally in Romanized form
* `source` (text): entity reporting the data; could be an organization, report, or document, generally in Romanized form
* `url` (text): web document corresponding to the `source` field
* `geolocation\_source` (text): attribution for geolocation information
* `wepp\_id` (text): a reference to a unique plant identifier in the widely-used PLATTS-WEPP database.
* `year\_of\_capacity\_data` (number): year the capacity information was reported
* `generation\_gwh\_2013` (number): electricity generation in gigawatt-hours reported for the year 2013
* `generation\_gwh\_2014` (number): electricity generation in gigawatt-hours reported for the year 2014
* `generation\_gwh\_2015` (number): electricity generation in gigawatt-hours reported for the year 2015
* `generation\_gwh\_2016` (number): electricity generation in gigawatt-hours reported for the year 2016
* `generation\_gwh\_2017` (number): electricity generation in gigawatt-hours reported for the year 2017
* `generation\_gwh\_2018` (number): electricity generation in gigawatt-hours reported for the year 2018
* `generation\_gwh\_2019` (number): electricity generation in gigawatt-hours reported for the year 2019
* `generation\_data\_source` (text): attribution for the reported generation information
* `estimated\_generation\_gwh\_2013` (number): estimated electricity generation in gigawatt-hours for the year 2013
* `estimated\_generation\_gwh\_2014` (number): estimated electricity generation in gigawatt-hours for the year 2014
* `estimated\_generation\_gwh\_2015` (number): estimated electricity generation in gigawatt-hours for the year 2015
* `estimated\_generation\_gwh\_2016` (number): estimated electricity generation in gigawatt-hours for the year 2016
* `estimated\_generation\_gwh\_2017` (number): estimated electricity generation in gigawatt-hours for the year 2017
* 'estimated\_generation\_note\_2013` (text): label of the model/method used to estimate generation for the year 2013
* `estimated\_generation\_note\_2014` (text): label of the model/method used to estimate generation for the year 2014
* `estimated\_generation\_note\_2015` (text): label of the model/method used to estimate generation for the year 2015
* `estimated\_generation\_note\_2016` (text): label of the model/method used to estimate generation for the year 2016
* `estimated\_generation\_note\_2017` (text): label of the model/method used to estimate generation for the year 2017

**3. Exploratry Data Analysis (EDA) - Concluding Remarks**

Through exploratory data analysis, we gained insights into the dataset's characteristics:

* **Correlation between Features**: Numeric columns were examined for correlation, and highly correlated columns were flagged for possible removal.
* **Imbalanced Classes**: The fuel type categories were not uniformly distributed, which required careful consideration during model training.
* **Missing Data**: Missing values, especially in the geolocation and capacity columns, needed to be handled properly to avoid data leakage or biased results.

**4. Pre-processing Pipeline**

To prepare the dataset for machine learning models, we designed a robust data preprocessing pipeline:

**4.1 Handling Missing Values**

* **Numeric Columns**: Missing values in numeric columns, such as capacity\_mw and latitude, were filled with the **mean** of each column.
* **Categorical Columns**: For categorical variables, such as primary\_fuel, missing values were filled with the **mode** (the most frequent value).

python

Copy code

data.fillna(data.mean(numeric\_only=True), inplace=True)

for col in data.select\_dtypes(include=[object]).columns:

data[col].fillna(data[col].mode()[0], inplace=True)

**4.2 Feature Engineering**

* **Dropping Irrelevant Columns**: Certain columns, such as gppd\_idnr (an identifier), name, owner, url, and other metadata, were dropped, as they do not provide meaningful information for the prediction tasks.
* **Handling Multicollinearity**: Highly correlated features can negatively affect the performance of machine learning models, especially in regression tasks. We calculated the correlation matrix and removed columns with correlation values above a threshold (0.85).

python

Copy code

threshold = 0.85

numeric\_data = data.select\_dtypes(include=[np.number])

corr\_matrix = numeric\_data.corr().abs()

upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

to\_drop = [column for column in upper.columns if any(upper[column] > threshold)]

data = data.drop(columns=to\_drop, axis=1)

**4.3 Encoding Categorical Variables**

We used **Label Encoding** to convert the categorical target variable (primary\_fuel) into numeric format. This process maps each unique fuel type to a numerical label.

python

Copy code

label\_enc = LabelEncoder()

data['primary\_fuel'] = label\_enc.fit\_transform(data['primary\_fuel'])

**4.4 Splitting the Data**

For the two separate prediction tasks, we split the dataset into features (X) and targets (y). We used train\_test\_split to divide the data into training and testing sets for both classification and regression tasks.

python

Copy code

X = data.drop(['primary\_fuel', 'capacity\_mw'], axis=1)

y\_classification = data['primary\_fuel']

y\_regression = data['capacity\_mw']

X\_train\_class, X\_test\_class, y\_train\_class, y\_test\_class = train\_test\_split(X, y\_classification, test\_size=0.3, random\_state=42)

X\_train\_reg, X\_test\_reg, y\_train\_reg, y\_test\_reg = train\_test\_split(X, y\_regression, test\_size=0.3, random\_state=42)

**4.5 Feature Scaling**

Since we were using machine learning algorithms like **Random Forest** (for classification) and **Linear Regression** (for capacity prediction), scaling the numeric features was important. We applied **StandardScaler** to ensure the features had a mean of 0 and a standard deviation of 1.

python

Copy code

scaler = StandardScaler()

X\_train\_class\_scaled = scaler.fit\_transform(X\_train\_class.select\_dtypes(include=[np.number]))

X\_test\_class\_scaled = scaler.transform(X\_test\_class.select\_dtypes(include=[np.number]))

X\_train\_reg\_scaled = scaler.fit\_transform(X\_train\_reg.select\_dtypes(include=[np.number]))

X\_test\_reg\_scaled = scaler.transform(X\_test\_reg.select\_dtypes(include=[np.number]))

**5. Building Machine Learning Models**

simple definitions for **classification** and **regression**:

* **Classification**: A type of supervised machine learning where the goal is to categorize data into distinct classes or categories. For example, classifying emails as "spam" or "not spam."
* **Regression**: A type of supervised machine learning where the goal is to predict a continuous numerical value based on input data. For example, predicting the price of a house based on its features like size and location.

Both techniques are used for prediction, but classification deals with discrete outcomes (labels) while regression deals with continuous outcomes (numeric values).

**5.1 Classification: Predicting Primary Fuel Type**

For the classification task, we used a **Random Forest Classifier**. Random Forests are known for their robustness and ability to handle imbalanced datasets effectively.

python

Copy code

from sklearn.ensemble import RandomForestClassifier

rf\_classifier = RandomForestClassifier()

rf\_classifier.fit(X\_train\_class\_scaled, y\_train\_class)

y\_pred\_class = rf\_classifier.predict(X\_test\_class\_scaled)

**Model Evaluation: Classification**

We evaluated the model using a **classification report** and **accuracy score**. The classification report includes precision, recall, and F1-score for each fuel type, and accuracy provides an overall performance measure.

python

Copy code

from sklearn.metrics import classification\_report, accuracy\_score

print("Classification Report:\n", classification\_report(y\_test\_class, y\_pred\_class))

print("Accuracy:", accuracy\_score(y\_test\_class, y\_pred\_class))

The initial accuracy was around 75%, which indicated that the model could predict fuel types with reasonable accuracy. However, there was room for improvement through hyperparameter tuning.

**5.2 Regression: Predicting Capacity (MW)**

For the regression task, we used a **Linear Regression** model to predict the capacity of power plants in megawatts.

python

Copy code

from sklearn.linear\_model import LinearRegression

lr\_regressor = LinearRegression()

lr\_regressor.fit(X\_train\_reg\_scaled, y\_train\_reg)

y\_pred\_reg = lr\_regressor.predict(X\_test\_reg\_scaled)

**Model Evaluation: Regression**

We evaluated the regression model using **Mean Squared Error (MSE)** and **R-squared** metrics. The R-squared value indicates how well the model explains the variance in the target variable.

python

Copy code

from sklearn.metrics import mean\_squared\_error, r2\_score

print("Mean Squared Error:", mean\_squared\_error(y\_test\_reg, y\_pred\_reg))

print("R-squared:", r2\_score(y\_test\_reg, y\_pred\_reg))

The initial R-squared score was 0.55, which suggested the model explained about 55% of the variance in the data.

**6. Hyperparameter Tuning**

**6.1 Tuning the Random Forest Classifier**

To improve the performance of the **Random Forest Classifier**, we performed hyperparameter tuning using **GridSearchCV**. We tuned parameters such as the number of estimators, the maximum depth of trees, and the minimum samples required to split a node.

python

Copy code

from sklearn.model\_selection import GridSearchCV

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

}

grid\_search\_rf = GridSearchCV(RandomForestClassifier(), param\_grid, cv=5, verbose=2)

grid\_search\_rf.fit(X\_train\_class\_scaled, y\_train\_class)

**6.2 Results**

After tuning the hyperparameters, the **Random Forest Classifier** achieved an accuracy of approximately 80%. The precision and recall for minority classes, such as **wind** and **solar**, also improved.

**7. Conclusion**

This project successfully built machine learning models to predict the primary fuel type and capacity of global power plants. We achieved:

* **80% accuracy** in predicting the primary fuel type using a Random Forest Classifier.
* A reasonable **R-squared score** of 0.55 in predicting plant capacity (in MW) using Linear Regression.

Further improvements could involve using more advanced algorithms, such as **XGBoost** or **Neural Networks**, to boost performance, as well as more rigorous feature engineering. The models developed here provide a solid foundation for automating the analysis of power plant characteristics.