# ADS\_phase 3

APPLIED DATA SCIENCE – ELECTRICITY PRICES PREDICTION

**Phase 3: Development Part 1**

**1.DATASET:**

**Dataset gathering:**

Electricity prices prediction is going to perform in this project so datas are gathered from kaggle site

Dataset link : <https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction>

**Implementation:**

dataset downloaded from kaggle site as a zip file and saved to a required directory , the data set zip is extracted and get a data set in csv file (electricity.csv)

**dataset explanation:**

dataset has 18 columns and 38,015 rows of datas

columns contains : DateTime, Holiday, HolidayFlag, DayOfWeek, WeekOfYear, Day, Month, Year, PeriodOfDay, ForecastWindProduction, SystemLoadEA, SMPEA, ORKTemperature

Rows contains: 1,2,3……38014,38015 and columns data

**2.Import the required libraries:**

to import libraries:

import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split

**3.Importing the data set(read data set; create matrix)**

To import the data set and read the dataset and create matrix :

import pandas as pd

# Assuming dataset is named 'Electricity.csv'

data = pd.read\_csv('Electricity.csv')

# Create a matrix of the dataset (assuming you want to work with a numerical matrix)

matrix = data.values

**4.Handling the Missing Data:**

handling missing data using the **SimpleImputer** class from the **sklearn.preprocessing,** the **SimpleImputer** class from the **sklearn.preprocessing** library to handle missing data in your dataset. The **SimpleImputer** allows you to fill in missing values with a chosen strategy.

To handle the missing data use:

from sklearn.impute import SimpleImputer

# Initialize the SimpleImputer with your chosen strategy (e.g., mean)

imputer = SimpleImputer(strategy='mean')

# Assuming 'data' is dataset and 'column\_with\_missing\_values' is the column with missing data

data['column\_with\_missing\_values'] = imputer.fit\_transform(data[['column\_with\_missing\_values']])

**5. Encoding Categorical Data.(one-hot encoding)**

If dataset contains categorical columns that need to be one-hot encoded, use the OneHotEncoder from sklearn. Replace 'categorical\_column' with the actual name of the categorical column.

To perform encoder :

encoder = OneHotEncoder(sparse=False)data\_encoded = pd.DataFrame(encoder.fit\_transform(data[['categorical\_column']]))data = pd.concat([data, data\_encoded], axis=1)

**6.Splitting the data set into test set and training set:**

To split your dataset into a training set and a test set, you can use the **train\_test\_split** function from the **sklearn.model\_selection,** Assuming you have features stored in a variable **X** and the target variable in a variable **Y**, you can split the dataset into training and test sets

* Import the **train\_test\_split** function from **sklearn.model\_selection**.
* **X** should contain your features, and **Y** should contain your target variable.
* The **test\_size** parameter controls the proportion of the dataset that goes into the test set (e.g., 0.2 means 20% of the data for testing).
* The **random\_state** parameter is used to ensure reproducibility. It sets a seed for the random number generator.

To perform test and train :

from sklearn.model\_selection import train\_test\_split

# Split the dataset into training and test sets (adjust test\_size as needed)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

**7.Feature Scaling.(import StandardScaler)**

perform feature scaling using the **StandardScaler** from the **sklearn.preprocessing** library,

Feature Scaling:

Apply feature scaling to your features. This step helps to standardize the range of features, making them more suitable for many machine learning algorithms

* Import the **StandardScaler** from **sklearn.preprocessing**.
* Create an instance of the **StandardScaler**.
* Fit the scaler to your training data (X\_train) and transform it to scale the features.
* Transform the test data (X\_test) using the same scaler that was fitted to the training data.

To perform scalar :

from sklearn.preprocessing import StandardScaler

# Create an instance of the StandardScaler

scaler = StandardScaler()

# Fit the scaler to your training data and transform it

X\_train = scaler.fit\_transform(X\_train)

# Transform the test data using the same scaler

X\_test = scaler.transform(X\_test)

**Conclusion**

the provided steps are essential for preparing a dataset for analysis or machine learning. They involve importing libraries, loading the dataset, handling missing data, encoding categorical features, splitting the data into training and testing sets, and performing feature scaling. These steps are crucial for ensuring data quality and model performance.

Top of Form