# Handling Energy Consumption Data with Pandas: Missing Values and Data Preprocessing

## Objectives:

1.Handling Missing Values: We will demonstrate how to handle missing values in energy consumption data by removing rows/columns, imputing values using mean/median, applying forward/backward filling, and flagging missing data.

2.Data Preprocessing: We will normalize/standardize the data, encode categorical variables, and implement feature engineering for further analysis.

### Step 1: Import Pandas and Create a Dataset with Missing Values

We'll first import the necessary libraries and create a dataset that includes some missing values (NaN).

In [1]:
```
import pandas as pd
import numpy as np

# Sample data with missing values
data = {
    "Energy Source": ["Solar", "Wind", "Hydropower", "Geothermal", "Biomass"
    "Energy Consumption (MWh)": [1200, np.nan, 2900, np.nan, 2500, 3200],
    "Cost (Million $)": [200, 400, np.nan, 150, 250, np.nan]
}

# Create a DataFrame
energy_df = pd.DataFrame(data)

print("Original Energy Data with Missing Values:")
print(energy_df)
```

```
Original Energy Data with Missing Values:
  Energy Source  Energy Consumption (MWh)  Cost (Million $)
0         Solar                    1200.0             200.0
1          Wind                       NaN             400.0
2    Hydropower                    2900.0               NaN
3    Geothermal                       NaN             150.0
4       Biomass                    2500.0             250.0
5       Nuclear                    3200.0               NaN
```

We created a Pandas DataFrame energy_df representing energy sources and their consumption and costs.

The dataset includes some NaN (missing) values, which we will handle in the next steps.

# 1. Handling Missing Values

### 1.1. Remove Rows with Missing Values

We can remove rows that contain any missing values using dropna().

In [2]:
```
# Remove rows with any missing values
cleaned_df = energy_df.dropna()

print("\nData After Removing Rows with Missing Values:")
print(cleaned_df)
```

```
Data After Removing Rows with Missing Values:
  Energy Source  Energy Consumption (MWh)  Cost (Million $)
0         Solar                    1200.0             200.0
4       Biomass                    2500.0             250.0
```

The above code snippet removes rows where any column has missing data. This method is straightforward but may result in losing a significant amount of data.

### 1.2. Impute Missing Values with the Mean

Instead of removing rows, we can impute missing values by filling them with the mean value of the column.

In [3]:
```
# Impute missing values in 'Energy Consumption (MWh)' with the mean
energy_df["Energy Consumption (MWh)"].fillna(energy_df["Energy Consumption (

# Impute missing values in 'Cost (Million $)' with the mean
energy_df["Cost (Million $)"].fillna(energy_df["Cost (Million $)"].mean(), i

print("\nData After Imputing Missing Values with Mean:")
print(energy_df)
```

```
Data After Imputing Missing Values with Mean:
  Energy Source  Energy Consumption (MWh)  Cost (Million $)
0         Solar                    1200.0             200.0
1          Wind                    2450.0             400.0
2    Hydropower                    2900.0             250.0
3    Geothermal                    2450.0             150.0
4       Biomass                    2500.0             250.0
5       Nuclear                    3200.0             250.0
```

We used the mean imputation method to fill missing values in both the Energy Consumption (MWh) and Cost (Million $) columns, ensuring that we retain the dataset while handling missing values.

### 1.3. Forward/Backward Filling

Another method is forward filling, where missing values are replaced by the previous valid entry.

In [4]:
```
# Forward fill missing values
forward_filled_df = energy_df.fillna(method="ffill")

print("\nData After Forward Filling:")
print(forward_filled_df)
```

```
Data After Forward Filling:
  Energy Source  Energy Consumption (MWh)  Cost (Million $)
0         Solar                    1200.0             200.0
1          Wind                    2450.0             400.0
2    Hydropower                    2900.0             250.0
3    Geothermal                    2450.0             150.0
4       Biomass                    2500.0             250.0
5       Nuclear                    3200.0             250.0
```

Forward filling (ffill) replaces missing values with the previous non-missing value in the column, which is useful when data is time-series-based.

### 1.4. Flag Missing Values

We can also create a separate column to flag missing values before imputation.

In [5]:
```
# Create a flag column indicating missing values in 'Energy Consumption (MWh
energy_df["Missing Consumption"] = energy_df["Energy Consumption (MWh)"].isr

print("\nData with Missing Values Flagged:")
print(energy_df)
```

```
Data with Missing Values Flagged:
  Energy Source  Energy Consumption (MWh)  Cost (Million $)  \
0         Solar                    1200.0             200.0
1          Wind                    2450.0             400.0
2    Hydropower                    2900.0             250.0
3    Geothermal                    2450.0             150.0
4       Biomass                    2500.0             250.0
5       Nuclear                    3200.0             250.0

   Missing Consumption
0                    0
1                    0
2                    0
3                    0
4                    0
5                    0
```

The Missing Consumption column flags missing values with 1 (missing) or 0 (not missing), which helps track imputed values.

# 2. Data Preprocessing

### 2.1. Normalization (Min-Max Scaling)

We will scale the data to a range between 0 and 1 using Min-Max Scaling.

In [6]:
```
from sklearn.preprocessing import MinMaxScaler

# Normalize the 'Energy Consumption (MWh)' and 'Cost (Million $)'
scaler = MinMaxScaler()
energy_df[["Energy Consumption (MWh)", "Cost (Million $)"]] = scaler.fit_tra
    energy_df[["Energy Consumption (MWh)", "Cost (Million $)"]]
)

print("\nData After Normalization (Min-Max Scaling):")
print(energy_df)
```

```
Data After Normalization (Min-Max Scaling):
  Energy Source  Energy Consumption (MWh)  Cost (Million $)  \
0         Solar                     0.000               0.2
1          Wind                     0.625               1.0
2    Hydropower                     0.850               0.4
3    Geothermal                     0.625               0.0
4       Biomass                     0.650               0.4
5       Nuclear                     1.000               0.4

   Missing Consumption
0                    0
1                    0
2                    0
3                    0
4                    0
5                    0
```

Min-Max Scaling normalizes the energy consumption and cost values, scaling them to a range between 0 and 1. This is useful when comparing features with different units or magnitudes.

### 2.2. Standardization (Z-score Scaling)

Alternatively, we can use standardization, which centers the data around a mean of 0 with a standard deviation of 1.

In [7]:
```
from sklearn.preprocessing import StandardScaler

# Standardize the 'Energy Consumption (MWh)' and 'Cost (Million $)'
scaler = StandardScaler()
energy_df[["Energy Consumption (MWh)", "Cost (Million $)"]] = scaler.fit_tra
    energy_df[["Energy Consumption (MWh)", "Cost (Million $)"]]
)

print("\nData After Standardization (Z-score Scaling):")
print(energy_df)
```

```
Data After Standardization (Z-score Scaling):
  Energy Source  Energy Consumption (MWh)  Cost (Million $)  \
0         Solar             -2.005893e+00     -6.546537e-01
1          Wind              3.563181e-16      1.963961e+00
2    Hydropower              7.221213e-01      1.817029e-16
3    Geothermal              3.563181e-16     -1.309307e+00
4       Biomass              8.023570e-02      1.817029e-16
5       Nuclear              1.203536e+00      1.817029e-16

   Missing Consumption
0                    0
1                    0
2                    0
3                    0
4                    0
5                    0
```

Z-score scaling standardizes the values, making the mean 0 and standard deviation 1, which is useful when dealing with normally distributed data.

### 2.3. Encoding Categorical Variables

We'll convert the categorical column Energy Source into numeric format using one-hot encoding.

In [8]:
```
# One-hot encode the 'Energy Source' column
energy_encoded_df = pd.get_dummies(energy_df, columns=["Energy Source"])

print("\nData After One-Hot Encoding Categorical Variables:")
print(energy_encoded_df)
```

```
Data After One-Hot Encoding Categorical Variables:
   Energy Consumption (MWh)  Cost (Million $)  Missing Consumption  \
0             -2.005893e+00     -6.546537e-01                    0
1              3.563181e-16      1.963961e+00                    0
2              7.221213e-01      1.817029e-16                    0
3              3.563181e-16     -1.309307e+00                    0
4              8.023570e-02      1.817029e-16                    0
5              1.203536e+00      1.817029e-16                    0

   Energy Source_Biomass  Energy Source_Geothermal  Energy Source_Hydropow
er  \
0                      0                         0
0
1                      0                         0
0
2                      0                         0
1
3                      0                         1
0
4                      1                         0
0
5                      0                         0
0

   Energy Source_Nuclear  Energy Source_Solar  Energy Source_Wind
0                      0                    1                   0
1                      0                    0                   1
2                      0                    0                   0
3                      0                    0                   0
4                      0                    0                   0
5                      1                    0                   0
```

One-hot encoding converts the Energy Source column into multiple binary columns, each representing the presence (1) or absence (0) of a specific energy source.

### 2.4. Feature Engineering

We can create a new feature that represents the ratio of energy consumption to cost.

In [9]:
```
# Create a new feature: Energy Consumption per Million $
energy_encoded_df["Consumption per $Million"] = energy_encoded_df["Energy Co

print("\nData with New Feature (Consumption per $Million):")
print(energy_encoded_df)
```

```
Data with New Feature (Consumption per $Million):
   Energy Consumption (MWh)  Cost (Million $)  Missing Consumption  \
0             -2.005893e+00     -6.546537e-01                    0
1              3.563181e-16      1.963961e+00                    0
2              7.221213e-01      1.817029e-16                    0
3              3.563181e-16     -1.309307e+00                    0
4              8.023570e-02      1.817029e-16                    0
5              1.203536e+00      1.817029e-16                    0

   Energy Source_Biomass  Energy Source_Geothermal  Energy Source_Hydropow
er  \
0                      0                         0
0
1                      0                         0
0
2                      0                         0
1
3                      0                         1
0
4                      1                         0
0
5                      0                         0
0

   Energy Source_Nuclear  Energy Source_Solar  Energy Source_Wind  \
0                      0                    1                   0
1                      0                    0                   1
2                      0                    0                   0
3                      0                    0                   0
4                      0                    0                   0
5                      1                    0                   0

   Consumption per $Million
0              3.064052e+00
1              1.814283e-16
2              3.974187e+15
3             -2.721424e-16
4              4.415764e+14
5              6.623646e+15
```

This new feature, Consumption per $Million, calculates how much energy is produced per million dollars spent, providing insight into the efficiency of energy sources.

### Conclusion

In this lab assignment, we handled missing values in energy consumption data by:

```
Removing rows with missing values,
Imputing missing values with the mean,
Using forward filling, and
Flagging missing values.
```

We then applied data preprocessing techniques such as normalization, standardization, encoding categorical variables, and feature engineering to enhance the dataset. These