DEE

March 18, 2023

[1]:
```
import pandas as pd
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
from sklearn.metrics import r2_score
```

[2]:
```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
```

[3]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

[4]:
```
# to ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

0.0.1 With the help of Pandas, read the ".csv" file and performing some task

[5]:
```
data1 = pd.read_csv("dee.csv")
```

[6]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Hydroelectric  365 non-null    float64
 1   Nuclear        365 non-null    int64
 2   Coal           365 non-null    float64
 3   Fuel           365 non-null    float64
 4   Gas            365 non-null    float64
 5   Special        365 non-null    int64
```

```
 6   Consume        365 non-null    float64
dtypes: float64(5), int64(2)
memory usage: 20.1 KB
```

[7]:
```python
data1.head(4)
```

[7]:

	Hydroelectric	Nuclear	Coal	Fuel	Gas	Special	Consume
0	179183.0	175973	78429.1	4680.73	8117.13	8023	1.732800
1	206035.0	186774	79129.5	4342.43	5715.18	8159	1.583500
2	198435.0	180633	64465.2	4566.84	0.00	8215	1.505310
3	187029.0	171382	51913.4	5342.54	0.00	8346	0.955205

[8]:
```python
data1.tail(4)
```

[8]:

	Hydroelectric	Nuclear	Coal	Fuel	Gas	Special	Consume
361	139186.0	179138	122708.0	8628.45	6884.80	12382	2.01027
362	162721.0	174891	182417.0	16296.30	31574.60	15540	2.12432
363	109797.0	172697	96596.3	5803.86	3245.38	11801	2.06576
364	130802.0	179072	127273.0	5733.01	5344.81	12929	1.43698

[9]:
```python
data1.isnull().any()
```

[9]:
```
Hydroelectric    False
Nuclear          False
Coal             False
Fuel             False
Gas              False
Special          False
Consume          False
dtype: bool
```

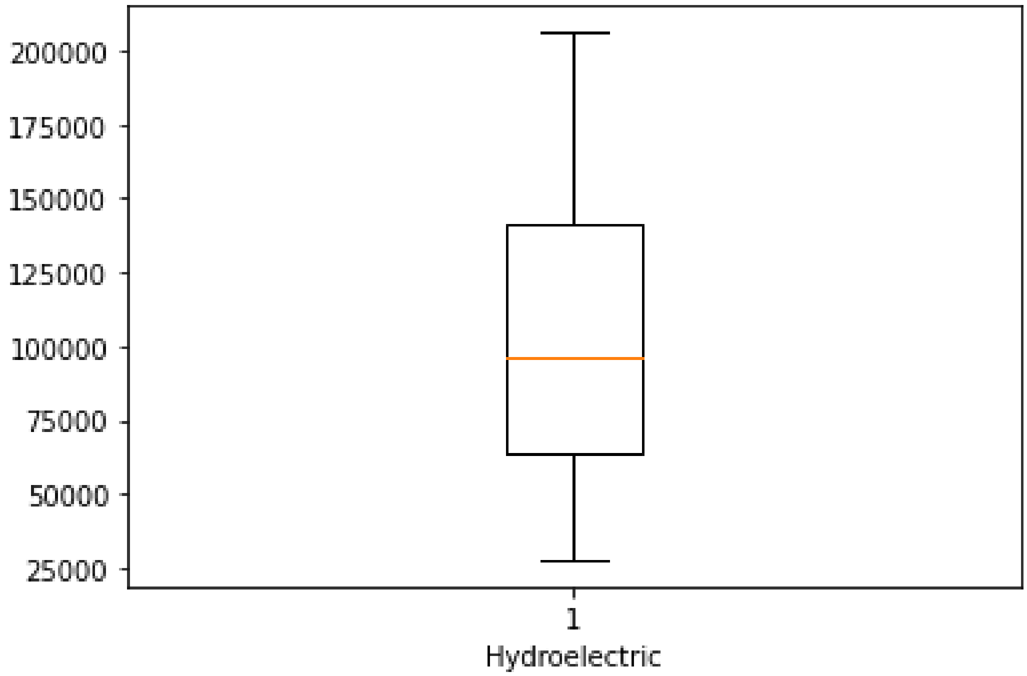
[10]:
```python
data1.isnull().sum()
```

[10]:
```
Hydroelectric    0
Nuclear          0
Coal             0
Fuel             0
Gas              0
Special          0
Consume          0
dtype: int64
```

[11]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
```

[12]:
```python
# Create a boxplot of the age variable
plt.boxplot(data1["Hydroelectric"])
plt.xlabel("Hydroelectric")
```

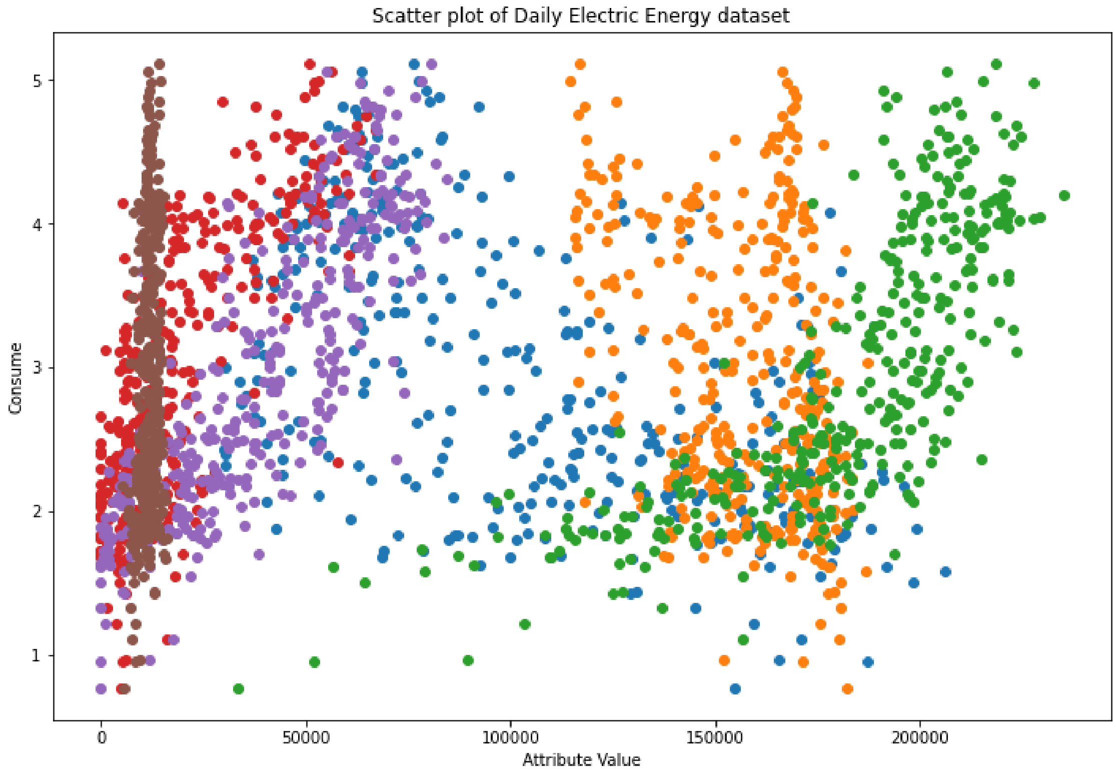
```
plt.show()
```



[13]:
```
fig, ax = plt.subplots(figsize=(12,8))

ax.scatter(data1["Hydroelectric"], data1["Consume"])
ax.scatter(data1["Nuclear"], data1["Consume"])
ax.scatter(data1["Coal"], data1["Consume"])
ax.scatter(data1["Fuel"], data1["Consume"])
ax.scatter(data1["Gas"], data1["Consume"])
ax.scatter(data1["Special"], data1["Consume"])

ax.set_xlabel("Attribute Value")
ax.set_ylabel("Consume")
ax.set_title("Scatter plot of Daily Electric Energy dataset")
plt.show()
```



[14]:
```python
X = data1.drop('Consume', axis=1).values
y = data1['Consume'].values
```

0.0.2 Create a linear regression model

[15]:
```python
linear_model = LinearRegression()
```

[16]:
```python
# Define the cross-validation method
cv = KFold(n_splits=5, shuffle=True, random_state=42)
```

[17]:
```python
# Evaluate the model using 5-fold cross-validation
linear_scores = cross_val_score(linear_model, X, y, cv=cv,␣
↪scoring='neg_mean_squared_error')
```

[18]:
```python
# Calculate the evaluation metrics from the scores
rmse = np.sqrt(-linear_scores.mean())
mse = -linear_scores.mean()
```

[19]:
```python
mae = -cross_val_score(linear_model, X, y, cv=cv,␣
↪scoring='neg_mean_absolute_error').mean()
r2 = cross_val_score(linear_model, X, y, cv=cv, scoring='r2').mean()
```

```
[20]: # Print the evaluation metrics for the Linear Regression model
print('Linear Regression Model:')
print('RMSE:', rmse)
print('MSE:', mse)
print('MAE:', mae)
print('R-squared:', r2)
```

```
Linear Regression Model:
RMSE: 0.41281996865485276
MSE: 0.17042032652019362
MAE: 0.31678813633724523
R-squared: 0.8159626900027581
```

```
[21]: linear_model.fit(X, y)
```

```
[21]: LinearRegression()
```

```
[22]: new_data = [[27881.8, 114760.0, 33537.0, 0.0, 0.0, 5307.0]]
prediction = linear_model.predict(new_data)

# Print the predicted value
print('Predicted value of Consume:', prediction[0])
```

```
Predicted value of Consume: 1.6479062344769242
```

0.0.3 Create an Artificial Neural Network model

```
[23]: # Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=42)
```

```
[24]: # Scale the features using standard scaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
[25]: # Create an Artificial Neural Network model
model = Sequential()
model.add(Dense(units=16, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(rate=0.2))
model.add(Dense(units=8, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
[26]: # Train the model
model.fit(X_train, y_train, epochs=100, batch_size=64, validation_split=0.2)
```

```
Epoch 1/100
4/4 [==============================] - 1s 83ms/step - loss: 11.9464 - val_loss:
9.0904
Epoch 2/100
4/4 [==============================] - 0s 11ms/step - loss: 11.6631 - val_loss:
8.8493
Epoch 3/100
4/4 [==============================] - 0s 11ms/step - loss: 11.5139 - val_loss:
8.6185
Epoch 4/100
4/4 [==============================] - 0s 12ms/step - loss: 11.1444 - val_loss:
8.4050
Epoch 5/100
4/4 [==============================] - 0s 11ms/step - loss: 11.0753 - val_loss:
8.1990
Epoch 6/100
4/4 [==============================] - 0s 12ms/step - loss: 10.5356 - val_loss:
8.0008
Epoch 7/100
4/4 [==============================] - 0s 11ms/step - loss: 10.2600 - val_loss:
7.8061
Epoch 8/100
4/4 [==============================] - 0s 12ms/step - loss: 10.2784 - val_loss:
7.6097
Epoch 9/100
4/4 [==============================] - 0s 12ms/step - loss: 9.9282 - val_loss:
7.4146
Epoch 10/100
4/4 [==============================] - 0s 11ms/step - loss: 9.5919 - val_loss:
7.2209
Epoch 11/100
4/4 [==============================] - 0s 11ms/step - loss: 9.4223 - val_loss:
7.0205
Epoch 12/100
4/4 [==============================] - 0s 10ms/step - loss: 9.2295 - val_loss:
6.8169
Epoch 13/100
4/4 [==============================] - 0s 11ms/step - loss: 9.0275 - val_loss:
6.6047
Epoch 14/100
4/4 [==============================] - 0s 12ms/step - loss: 8.7099 - val_loss:
6.3796
Epoch 15/100
4/4 [==============================] - 0s 12ms/step - loss: 8.3753 - val_loss:
6.1463
Epoch 16/100
4/4 [==============================] - 0s 11ms/step - loss: 8.2154 - val_loss:
5.9046
```

```
Epoch 17/100
4/4 [==============================] - 0s 11ms/step - loss: 7.9283 - val_loss:
5.6511
Epoch 18/100
4/4 [==============================] - 0s 13ms/step - loss: 7.4927 - val_loss:
5.3902
Epoch 19/100
4/4 [==============================] - 0s 12ms/step - loss: 7.0290 - val_loss:
5.1229
Epoch 20/100
4/4 [==============================] - 0s 12ms/step - loss: 6.9064 - val_loss:
4.8531
Epoch 21/100
4/4 [==============================] - 0s 12ms/step - loss: 6.5768 - val_loss:
4.5813
Epoch 22/100
4/4 [==============================] - 0s 15ms/step - loss: 5.9098 - val_loss:
4.2976
Epoch 23/100
4/4 [==============================] - 0s 11ms/step - loss: 5.8229 - val_loss:
4.0132
Epoch 24/100
4/4 [==============================] - 0s 11ms/step - loss: 5.5866 - val_loss:
3.7301
Epoch 25/100
4/4 [==============================] - 0s 10ms/step - loss: 5.2033 - val_loss:
3.4552
Epoch 26/100
4/4 [==============================] - 0s 10ms/step - loss: 4.7514 - val_loss:
3.1886
Epoch 27/100
4/4 [==============================] - 0s 12ms/step - loss: 4.3201 - val_loss:
2.9281
Epoch 28/100
4/4 [==============================] - 0s 11ms/step - loss: 4.3296 - val_loss:
2.6822
Epoch 29/100
4/4 [==============================] - 0s 12ms/step - loss: 3.9671 - val_loss:
2.4462
Epoch 30/100
4/4 [==============================] - 0s 11ms/step - loss: 3.5596 - val_loss:
2.2225
Epoch 31/100
4/4 [==============================] - 0s 11ms/step - loss: 3.2157 - val_loss:
2.0067
Epoch 32/100
4/4 [==============================] - 0s 10ms/step - loss: 3.0046 - val_loss:
1.8048
```

```
Epoch 33/100
4/4 [==============================] - 0s 10ms/step - loss: 2.5191 - val_loss:
1.6214
Epoch 34/100
4/4 [==============================] - 0s 11ms/step - loss: 2.4460 - val_loss:
1.4536
Epoch 35/100
4/4 [==============================] - 0s 11ms/step - loss: 2.5960 - val_loss:
1.3020
Epoch 36/100
4/4 [==============================] - 0s 13ms/step - loss: 2.5010 - val_loss:
1.1621
Epoch 37/100
4/4 [==============================] - 0s 10ms/step - loss: 1.8669 - val_loss:
1.0408
Epoch 38/100
4/4 [==============================] - 0s 10ms/step - loss: 2.0692 - val_loss:
0.9352
Epoch 39/100
4/4 [==============================] - 0s 11ms/step - loss: 1.7135 - val_loss:
0.8413
Epoch 40/100
4/4 [==============================] - 0s 11ms/step - loss: 2.0060 - val_loss:
0.7643
Epoch 41/100
4/4 [==============================] - 0s 11ms/step - loss: 1.5310 - val_loss:
0.7001
Epoch 42/100
4/4 [==============================] - 0s 10ms/step - loss: 1.7147 - val_loss:
0.6487
Epoch 43/100
4/4 [==============================] - 0s 11ms/step - loss: 1.6232 - val_loss:
0.6057
Epoch 44/100
4/4 [==============================] - 0s 11ms/step - loss: 1.7053 - val_loss:
0.5674
Epoch 45/100
4/4 [==============================] - 0s 10ms/step - loss: 1.5380 - val_loss:
0.5368
Epoch 46/100
4/4 [==============================] - 0s 11ms/step - loss: 1.5218 - val_loss:
0.5126
Epoch 47/100
4/4 [==============================] - 0s 12ms/step - loss: 1.6862 - val_loss:
0.4921
Epoch 48/100
4/4 [==============================] - 0s 12ms/step - loss: 1.5758 - val_loss:
0.4742
```

```
Epoch 49/100
4/4 [==============================] - 0s 11ms/step - loss: 1.5785 - val_loss:
0.4597
Epoch 50/100
4/4 [==============================] - 0s 11ms/step - loss: 1.4930 - val_loss:
0.4472
Epoch 51/100
4/4 [==============================] - 0s 10ms/step - loss: 1.4261 - val_loss:
0.4370
Epoch 52/100
4/4 [==============================] - 0s 12ms/step - loss: 1.4445 - val_loss:
0.4272
Epoch 53/100
4/4 [==============================] - 0s 11ms/step - loss: 1.4492 - val_loss:
0.4213
Epoch 54/100
4/4 [==============================] - 0s 11ms/step - loss: 1.3140 - val_loss:
0.4167
Epoch 55/100
4/4 [==============================] - 0s 10ms/step - loss: 1.4822 - val_loss:
0.4090
Epoch 56/100
4/4 [==============================] - 0s 11ms/step - loss: 1.4581 - val_loss:
0.4028
Epoch 57/100
4/4 [==============================] - 0s 11ms/step - loss: 1.3464 - val_loss:
0.3944
Epoch 58/100
4/4 [==============================] - 0s 10ms/step - loss: 1.4401 - val_loss:
0.3869
Epoch 59/100
4/4 [==============================] - 0s 10ms/step - loss: 1.3851 - val_loss:
0.3796
Epoch 60/100
4/4 [==============================] - 0s 10ms/step - loss: 1.3708 - val_loss:
0.3700
Epoch 61/100
4/4 [==============================] - 0s 10ms/step - loss: 1.6417 - val_loss:
0.3617
Epoch 62/100
4/4 [==============================] - 0s 10ms/step - loss: 1.3589 - val_loss:
0.3556
Epoch 63/100
4/4 [==============================] - 0s 11ms/step - loss: 1.2982 - val_loss:
0.3515
Epoch 64/100
4/4 [==============================] - 0s 10ms/step - loss: 1.3851 - val_loss:
0.3482
```

```
Epoch 65/100
4/4 [==============================] - 0s 10ms/step - loss: 1.5463 - val_loss:
0.3428
Epoch 66/100
4/4 [==============================] - 0s 11ms/step - loss: 1.4726 - val_loss:
0.3401
Epoch 67/100
4/4 [==============================] - 0s 11ms/step - loss: 1.0728 - val_loss:
0.3370
Epoch 68/100
4/4 [==============================] - 0s 12ms/step - loss: 1.2520 - val_loss:
0.3346
Epoch 69/100
4/4 [==============================] - 0s 10ms/step - loss: 1.2275 - val_loss:
0.3327
Epoch 70/100
4/4 [==============================] - 0s 11ms/step - loss: 1.4167 - val_loss:
0.3303
Epoch 71/100
4/4 [==============================] - 0s 10ms/step - loss: 1.1346 - val_loss:
0.3295
Epoch 72/100
4/4 [==============================] - 0s 11ms/step - loss: 1.3737 - val_loss:
0.3309
Epoch 73/100
4/4 [==============================] - 0s 10ms/step - loss: 1.1714 - val_loss:
0.3358
Epoch 74/100
4/4 [==============================] - 0s 11ms/step - loss: 1.2978 - val_loss:
0.3400
Epoch 75/100
4/4 [==============================] - 0s 13ms/step - loss: 1.4047 - val_loss:
0.3437
Epoch 76/100
4/4 [==============================] - 0s 13ms/step - loss: 1.0385 - val_loss:
0.3455
Epoch 77/100
4/4 [==============================] - 0s 12ms/step - loss: 1.3871 - val_loss:
0.3479
Epoch 78/100
4/4 [==============================] - 0s 13ms/step - loss: 1.2825 - val_loss:
0.3457
Epoch 79/100
4/4 [==============================] - 0s 21ms/step - loss: 1.2230 - val_loss:
0.3405
Epoch 80/100
4/4 [==============================] - 0s 12ms/step - loss: 1.1306 - val_loss:
0.3368
```

```
Epoch 81/100
4/4 [==============================] - 0s 11ms/step - loss: 0.9573 - val_loss:
0.3330
Epoch 82/100
4/4 [==============================] - 0s 11ms/step - loss: 1.0028 - val_loss:
0.3293
Epoch 83/100
4/4 [==============================] - 0s 13ms/step - loss: 1.2515 - val_loss:
0.3263
Epoch 84/100
4/4 [==============================] - 0s 11ms/step - loss: 1.1738 - val_loss:
0.3234
Epoch 85/100
4/4 [==============================] - 0s 11ms/step - loss: 1.2485 - val_loss:
0.3217
Epoch 86/100
4/4 [==============================] - 0s 11ms/step - loss: 1.4123 - val_loss:
0.3206
Epoch 87/100
4/4 [==============================] - 0s 10ms/step - loss: 1.0697 - val_loss:
0.3158
Epoch 88/100
4/4 [==============================] - 0s 10ms/step - loss: 1.1360 - val_loss:
0.3105
Epoch 89/100
4/4 [==============================] - 0s 10ms/step - loss: 1.0149 - val_loss:
0.3055
Epoch 90/100
4/4 [==============================] - 0s 10ms/step - loss: 1.0986 - val_loss:
0.3032
Epoch 91/100
4/4 [==============================] - 0s 11ms/step - loss: 1.0665 - val_loss:
0.3019
Epoch 92/100
4/4 [==============================] - 0s 12ms/step - loss: 1.1981 - val_loss:
0.3013
Epoch 93/100
4/4 [==============================] - 0s 11ms/step - loss: 0.8736 - val_loss:
0.2972
Epoch 94/100
4/4 [==============================] - 0s 11ms/step - loss: 1.0582 - val_loss:
0.2932
Epoch 95/100
4/4 [==============================] - 0s 13ms/step - loss: 1.1480 - val_loss:
0.2907
Epoch 96/100
4/4 [==============================] - 0s 10ms/step - loss: 1.2125 - val_loss:
0.2896
```

```
Epoch 97/100
4/4 [==============================] - 0s 10ms/step - loss: 1.0414 - val_loss:
0.2879
Epoch 98/100
4/4 [==============================] - 0s 11ms/step - loss: 1.2164 - val_loss:
0.2870
Epoch 99/100
4/4 [==============================] - 0s 11ms/step - loss: 1.1396 - val_loss:
0.2891
Epoch 100/100
4/4 [==============================] - 0s 10ms/step - loss: 1.1171 - val_loss:
0.2903
```

[26]:
```
<keras.callbacks.History at 0x2b6f7bb5f30>
```

[27]:
```
# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
3/3 [==============================] - 0s 1ms/step
```

[28]:
```
# Calculate the evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mse = mean_squared_error(y_test, y_pred)
```

[29]:
```
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

[30]:
```
# Print the evaluation metrics for the Artificial Neural Network model
print('Artificial Neural Network Model:')
print('RMSE:', rmse)
print('MSE:', mse)
print('MAE:', mae)
print('R-squared:', r2)
```

```
Artificial Neural Network Model:
RMSE: 0.5472146718518526
MSE: 0.2994438970899307
MAE: 0.4252916363822205
R-squared: 0.653178329338943
```

0.0.4 Create an XGBoost Regression model

[31]:
```
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
```

[32]:
```
# Evaluate the model using 5-fold cross-validation
xgb_scores = cross_val_score(xgb_model, X, y, cv=cv,␣
 ↪scoring='neg_mean_squared_error')
```

[33]:
```
# Calculate the evaluation metrics from the scores
xgb_rmse = np.sqrt(-xgb_scores.mean())
xgb_mse = -xgb_scores.mean()
```

[34]:
```
xgb_mae = -cross_val_score(xgb_model, X, y, cv=cv,␣
 ↪scoring='neg_mean_absolute_error').mean()
xgb_r2 = cross_val_score(xgb_model, X, y, cv=cv, scoring='r2').mean()
```

[35]:
```
# Print the evaluation metrics for the XGBoost Regression model
print('XGBoost Regression Model:')
print('RMSE:', xgb_rmse)
print('MSE:', xgb_mse)
print('MAE:', xgb_mae)
print('R-squared:', xgb_r2)
```

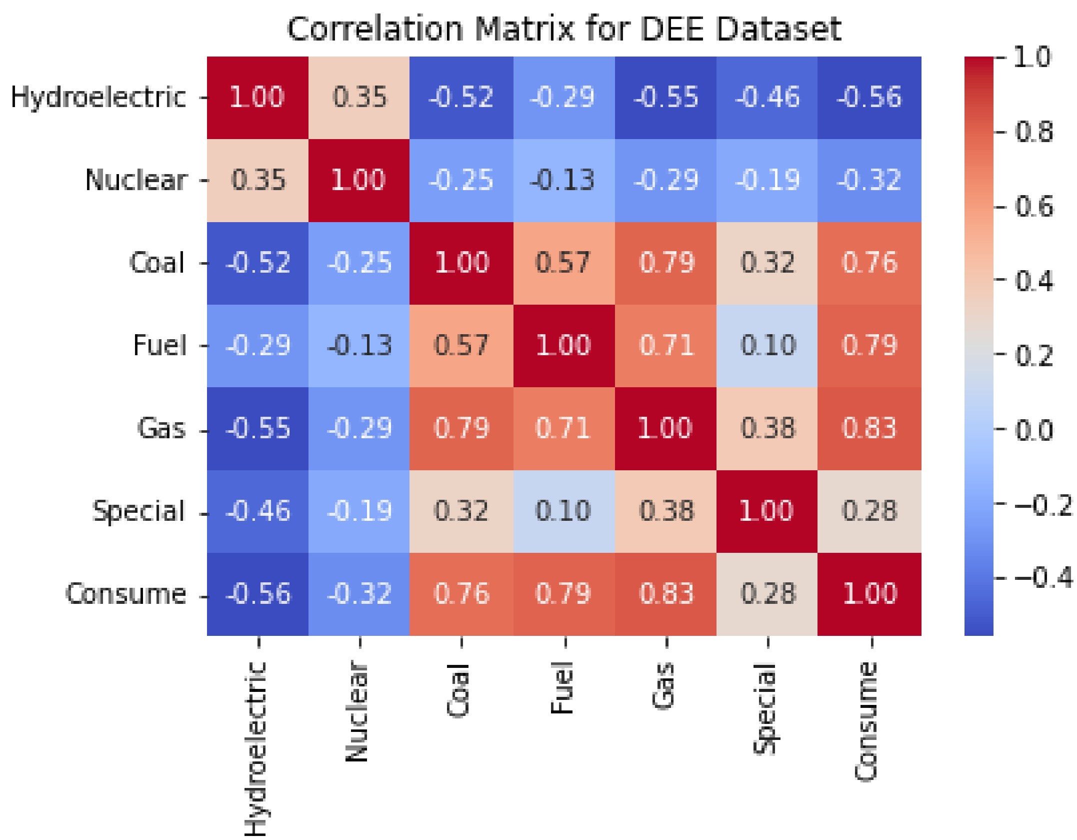
```
XGBoost Regression Model:
RMSE: 0.4554426365691612
MSE: 0.20742799520506905
MAE: 0.3363558875635225
R-squared: 0.7753485901175845
```

0.0.5 Visualization

[36]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

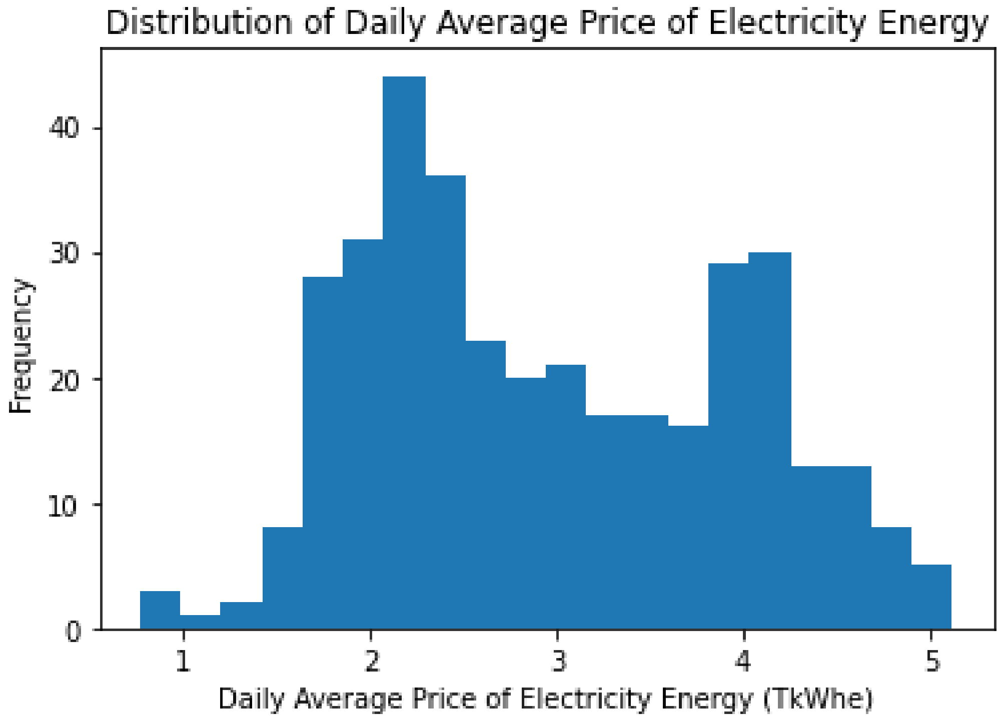
[37]:
```
# Compute the correlation matrix
corr = data1.corr()

# Plot the correlation matrix as a heatmap
sns.heatmap(corr, cmap='coolwarm', annot=True, fmt='.2f')
plt.title('Correlation Matrix for DEE Dataset')
plt.show()
```



[38]:

```
# Plot the distribution of the target variable "Consume"
plt.hist(data1['Consume'], bins=20)
plt.xlabel('Daily Average Price of Electricity Energy (TkWhe)')
plt.ylabel('Frequency')
plt.title('Distribution of Daily Average Price of Electricity Energy')
plt.show()
```


Distribution of Daily Average Price of Electricity Energy
Frequency
0
10
20
30
40
1
2
3
4
5
Daily Average Price of Electricity Energy (TkWhe)