EnerSustain

Saudi Arabia Energy Consumption Study

Prince Sattam bin Abdulaziz University

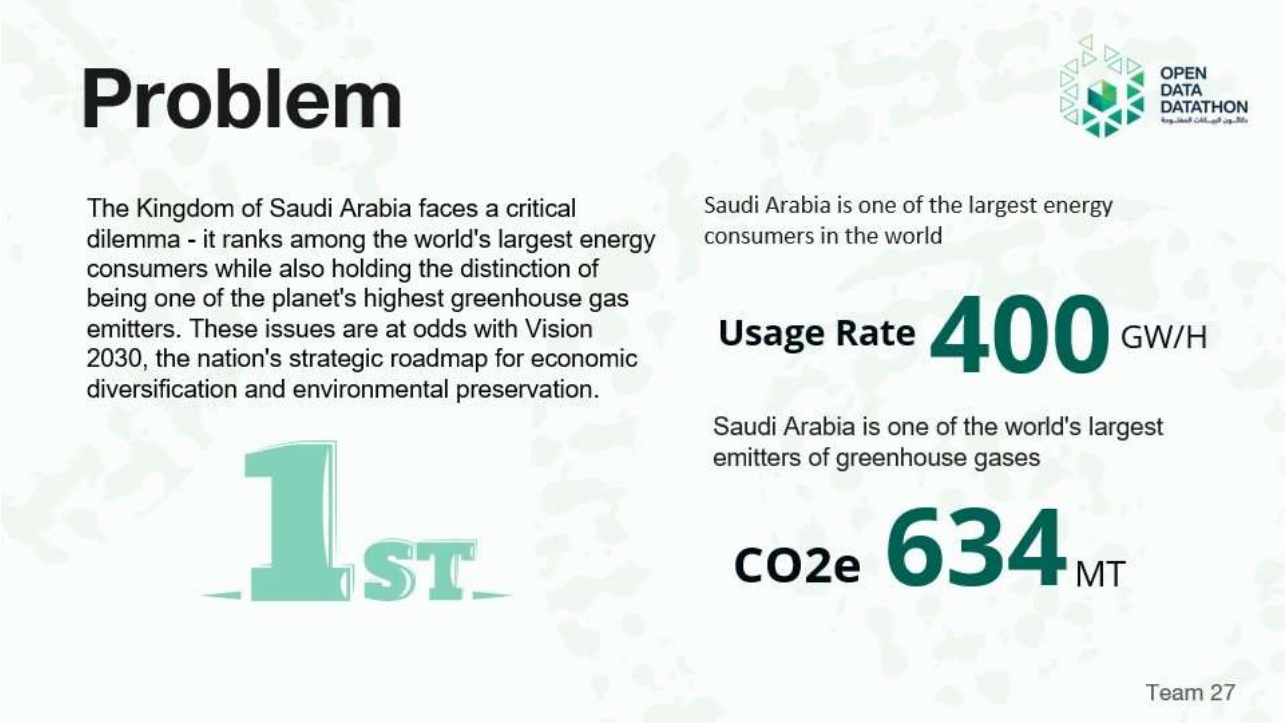
College of Computer Engineering and Sciences

Information Systems department

IS4811 | Data Science Fundamental

Supervisor: Dr.Fatma Al-Masmoudi - Dr.Fahdah Al-Marshad

Problem Statment:



Ideation

Introduction

Saudi Arabia's Vision 2030 is a comprehensive plan to diversify the country's economy and reduce its reliance on oil. One of the key goals of Vision 2030 is to increase the use of renewable energy and improve energy efficiency.

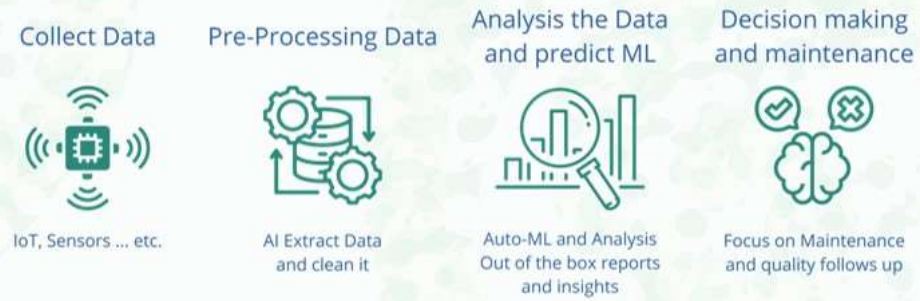
Decrease the Usage Rate **30%**

Increase Use of renewable energy **50%**

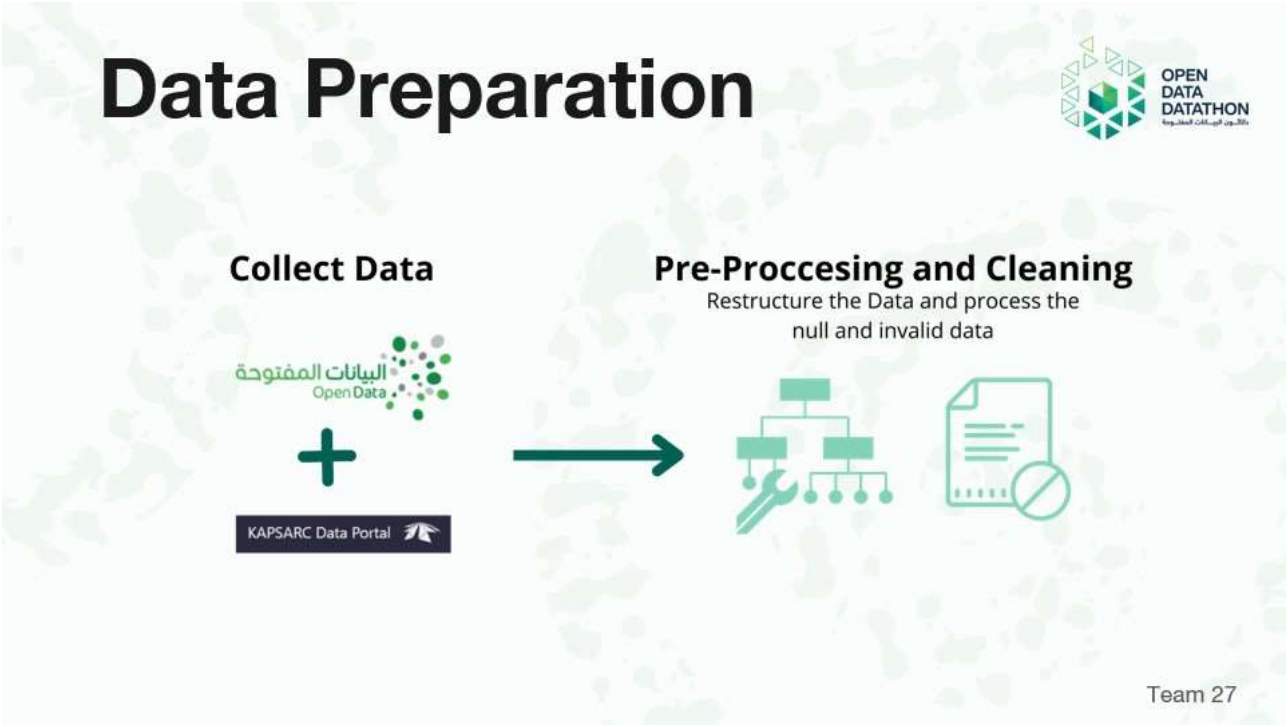
Team 27

EnerSustain

Data-Driven Forecasting for Sustainable Energy Management in KSA



Team 27




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

```
Mounted at /content/drive/
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns  # Import seaborn
!pip install seaborn
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.23.5)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.1.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.4
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.1.
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->
```

```
Electricity = pd.read_excel('/content/drive/MyDrive/Datasets/dataset1.xlsx',sheet_name='Sheet1')
Water = pd.read_excel('/content/drive/MyDrive/Datasets/dataset1.xlsx',sheet_name='Sheet2')
Gas = pd.read_excel('/content/drive/MyDrive/Datasets/dataset1.xlsx',sheet_name='Sheet3')
Sun = pd.read_excel('/content/drive/MyDrive/Datasets/dataset1.xlsx',sheet_name='Sheet4')
WaterU = pd.read_excel('/content/drive/MyDrive/Datasets/dataset1.xlsx',sheet_name='Sheet5')

print(Electricity,Water,WaterU,Gas,Sun)
```

```
      Year    Region        Sector Electricity Consumption Date_Object  \
0     2005   Central   Residential                24491767  2005-01-01
1     2005  Southern   Residential                 8320650  2005-01-01
2     2005  Southern    Industrial                  105349  2005-01-01
3     2005  Southern  Governmental                 1221641  2005-01-01
4     2005   Central    Commercial                 5039139  2005-01-01
..     ...       ...           ...                     ...         ...
403   2020   Eastern        Others              4605585،83  2020-01-01
404   2020   Western        Others             6526330،174  2020-01-01
405   2021   Eastern  Governmental             6882475،473  2021-01-01
406   2021  Southern  Governmental             6099399،827  2021-01-01
407   2021  Southern        Others             1256649،886  2021-01-01
```

```
    Periodicity
0       Annualy
1       Annualy
2       Annualy
3       Annualy
4       Annualy
..          ...
403     Annualy
404     Annualy
405     Annualy
406     Annualy
407     Annualy

[408 rows x 6 columns]      Date Frequency                 Indicator    Plant Name       Value
0    2005    Annual  Water Production by Plant     Satellite     9443579
1    2005    Annual  Water Production by Plant       Shuqaiq    37536366
2    2006    Annual      Water Export by Plant         Yanbu   112169072
3    2006    Annual      Water Export by Plant      Shoaibah   220837378
4    2007    Annual       Water Export by SWCC          SWCC  1066827984
..    ...       ...                        ...           ...         ...
250  2021    Annual  Water production by plant      Shoaibah   370100000
251  2017    Annual  Water Production by Plant        Jubail   431618438
252  2017    Annual  Water Production by Plant        Jeddah   196785920
253  2017    Annual  Water Production by Plant        Khafji     7012375
254  2016    Annual  Water Production by Plant  Ras Al-Khair   306150899

[255 rows x 5 columns]                  Region  Year  Value water
0                   Riyadh  2015          337
1                   Riyadh  2017          357
2                   Makkah  2012          241
3                   Makkah  2015          237
4                  Madinah  2010          232
..                     ...   ...          ...
135              Al-Bahah  2014          153
136               Al-Jouf  2011          240
137               Al-Jouf  2015          222
138               Al-Jouf  2017          149
139  Kingdom-wide average  2015          263

[140 rows x 3 columns]     Year           Region Butane LPG Kerosene Diesel
0   2017   Eastern Region      77,09     0,58   0,55
1   2017           Riyadh       91,9      0,9   0,08
2   2017           Makkah      97,19     0,07    0,1
3   2017          Madinah      91,91      0,5   0,09
4   2017           Qaseem      98,98     0,87   0,05
```

```
Electricity.shape
```

```
(408, 6)
```

```
Water.shape
```

```
(255, 5)
```

```
Gas.shape
```

```
(52, 5)
```

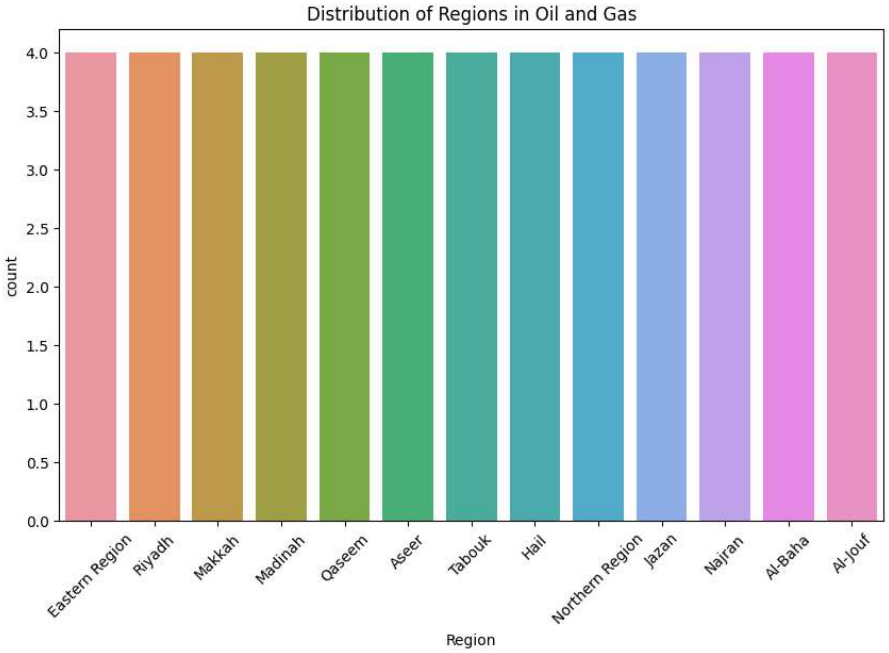
```
Sun.shape
```

```
(52, 3)
```

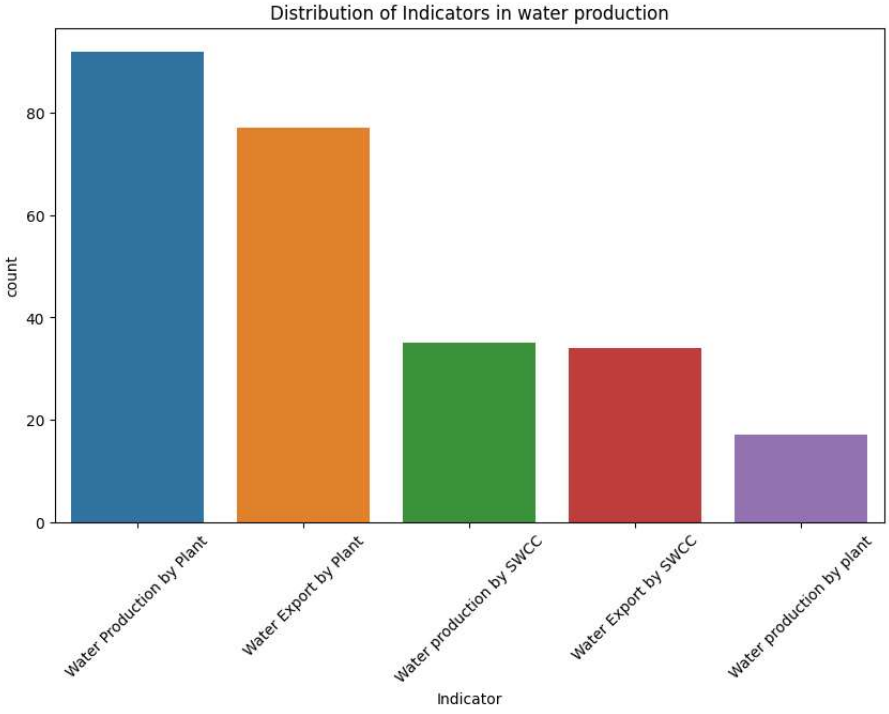
```
WaterU.shape
```

```
(140, 3)
```

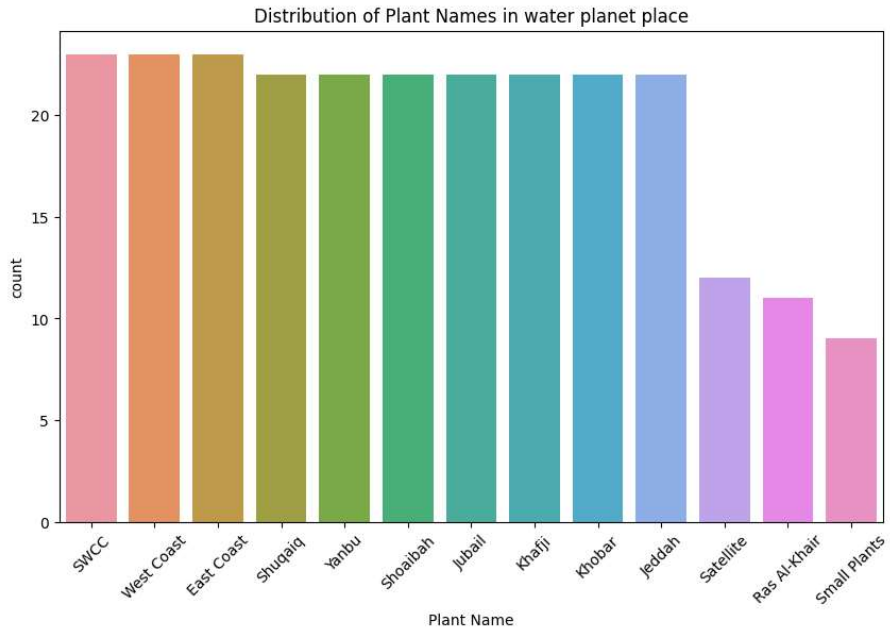
```
plt.figure(figsize=(10, 6))
sns.countplot(data=Gas, x='Region')  # Correct the syntax here
plt.title('Distribution of Regions in Oil and Gas')
plt.xticks(rotation=45)
plt.show()
```



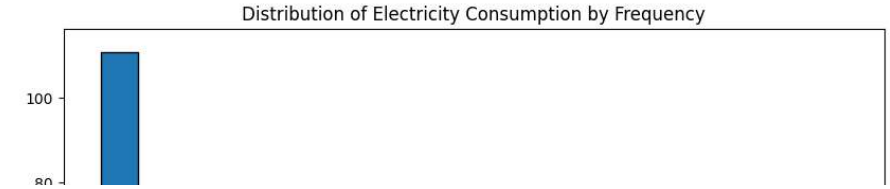

```
# Visualize the distribution of the 'Indicator' categorical variable in dataset 3
plt.figure(figsize=(10, 6))
sns.countplot(data=Water, x='Indicator', order=Water['Indicator'].value_counts().index)
plt.title('Distribution of Indicators in water production')
plt.xticks(rotation=45)
plt.show()
```



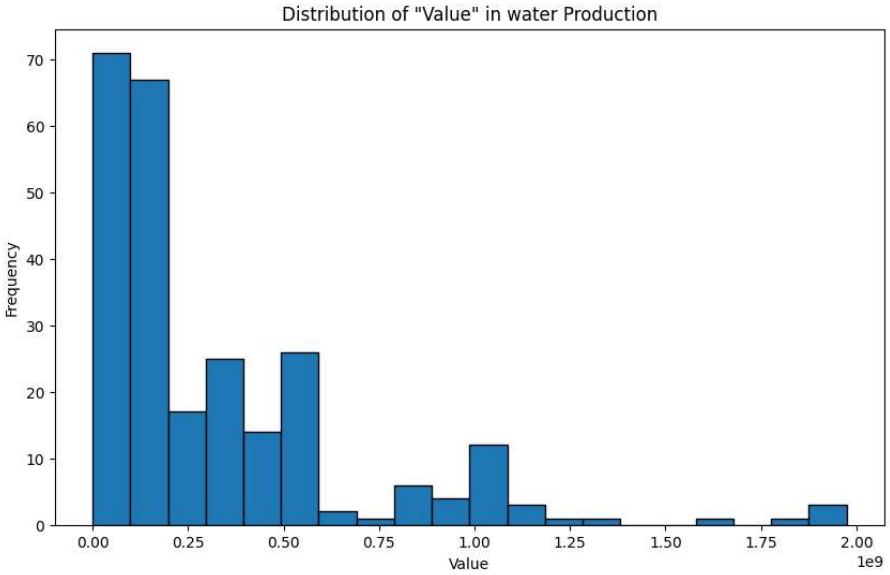
```
# Visualize the distribution of the 'Plant Name' categorical variable in dataset 4
plt.figure(figsize=(10, 6))
sns.countplot(data=Water, x='Plant Name', order=Water['Plant Name'].value_counts().index)
plt.title('Distribution of Plant Names in water planet place')
plt.xticks(rotation=45)
plt.show()
```



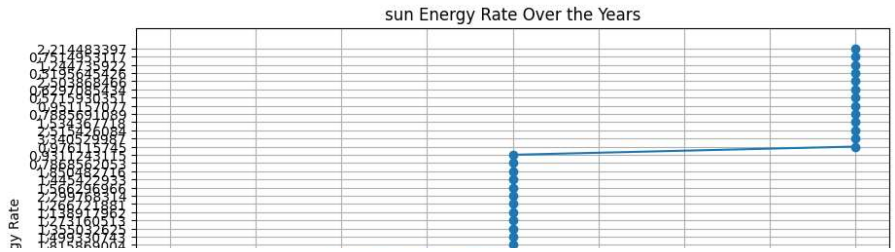

```
import matplotlib.pyplot as plt
import pandas as pd

Electricity['Electricity Consumption'] = pd.to_numeric(Electricity['Electricity Consumption'], errors='coerce')
Electricity = Electricity.dropna()  # Drop rows with missing values

plt.figure(figsize=(10, 6))
plt.hist(Electricity['Electricity Consumption'], bins=20, edgecolor='k')
plt.xlabel('Electricity Consumption')
plt.ylabel('Frequency')
plt.title('Distribution of Electricity Consumption by Frequency')
plt.show()
```




```
import matplotlib.pyplot as plt

# Example for dataset 2
plt.figure(figsize=(10, 6))
plt.hist(Water['Value'], bins=20, edgecolor='k')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Distribution of "Value" in water Production')
plt.show()
```

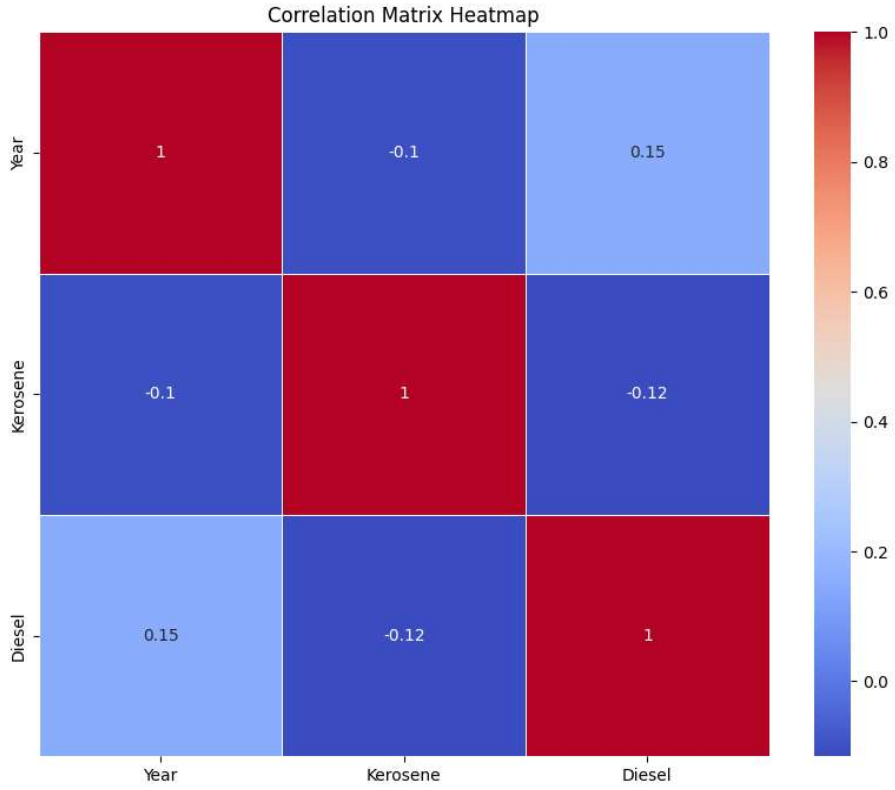



```
plt.figure(figsize=(10, 6))
plt.plot(Sun['Year'], Sun['sun energy rate'], marker='o')
plt.xlabel('Year')
plt.ylabel('sun Energy Rate')
plt.title('sun Energy Rate Over the Years')
plt.grid(True)
plt.show()
```



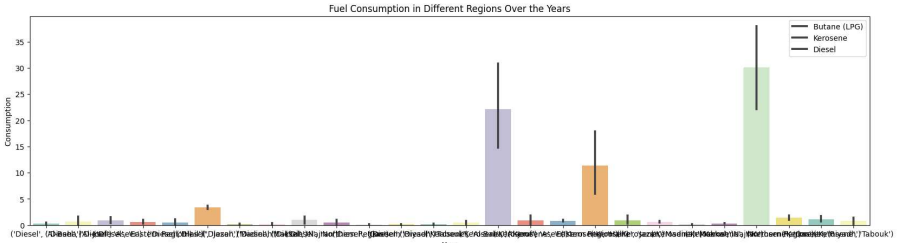

```
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = Gas.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
<ipython-input-18-08c25bc595f2>:3: FutureWarning: The default value of numeric_only in D
  correlation_matrix = Gas.corr()
```



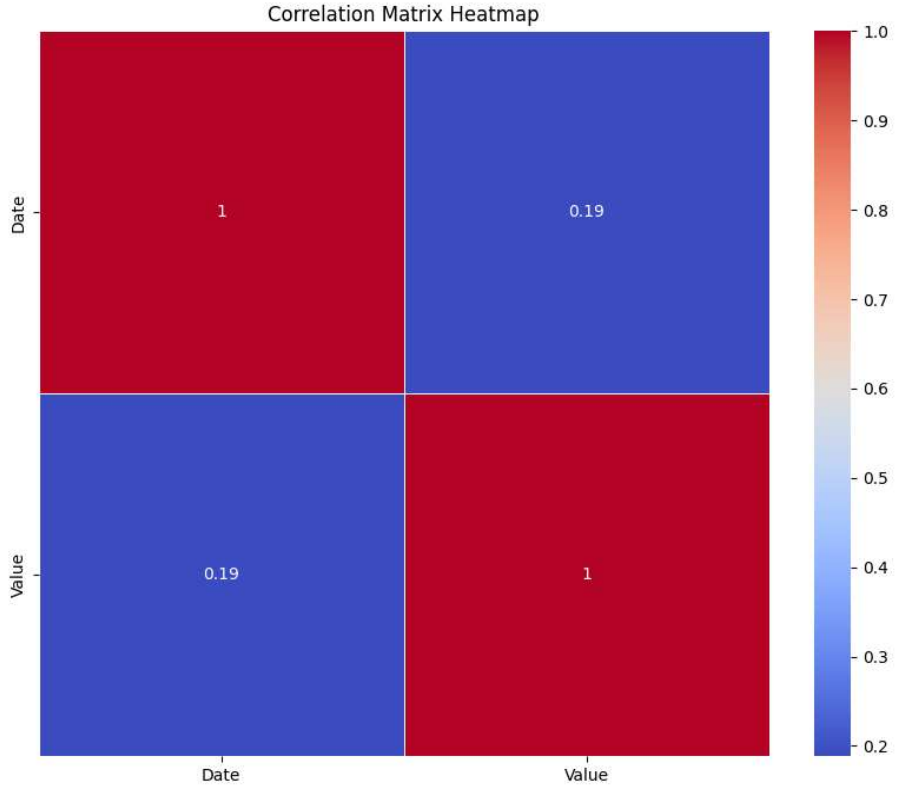

```
import matplotlib.pyplot as plt
import seaborn as sns

Gas_pivot = Gas.pivot_table(index='Year', columns='Region', values=['Butane LPG', 'Kerosene', 'Diesel'], aggfunc='sum')

plt.figure(figsize=(20, 5))
sns.barplot(data=Gas_pivot, palette='Set3')
plt.xlabel('Year')
plt.ylabel('Consumption')
plt.title('Fuel Consumption in Different Regions Over the Years')
plt.legend(['Butane LPG', 'Kerosene', 'Diesel'])
plt.show()
```

```
<ipython-input-43-b0b50a98be63>:4: FutureWarning: pivot_table dropped a column because i
  Gas_pivot = Gas.pivot_table(index='Year', columns='Region', values=['Butane LPG', 'Ker
```



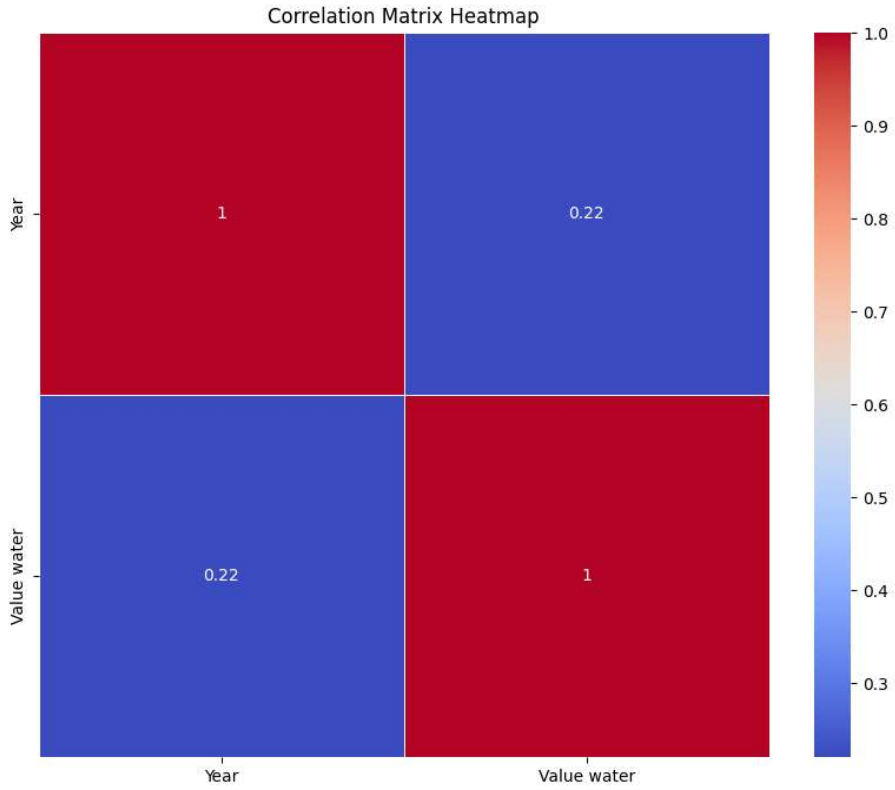

```
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = Water.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
<ipython-input-19-4a38f8e1dfae>:3: FutureWarning: The default value of numeric_only in D
  correlation_matrix = Water.corr()
```



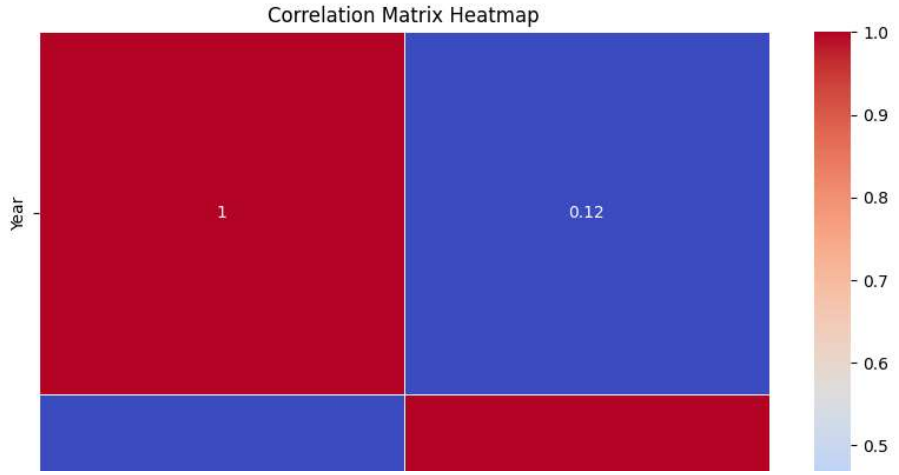
```
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = WaterU.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
<ipython-input-20-5e35bebd3fa8>:3: FutureWarning: The default value of numeric_only in D
  correlation_matrix = WaterU.corr()
```

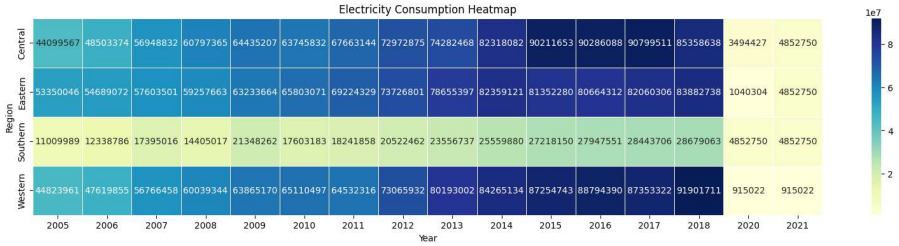


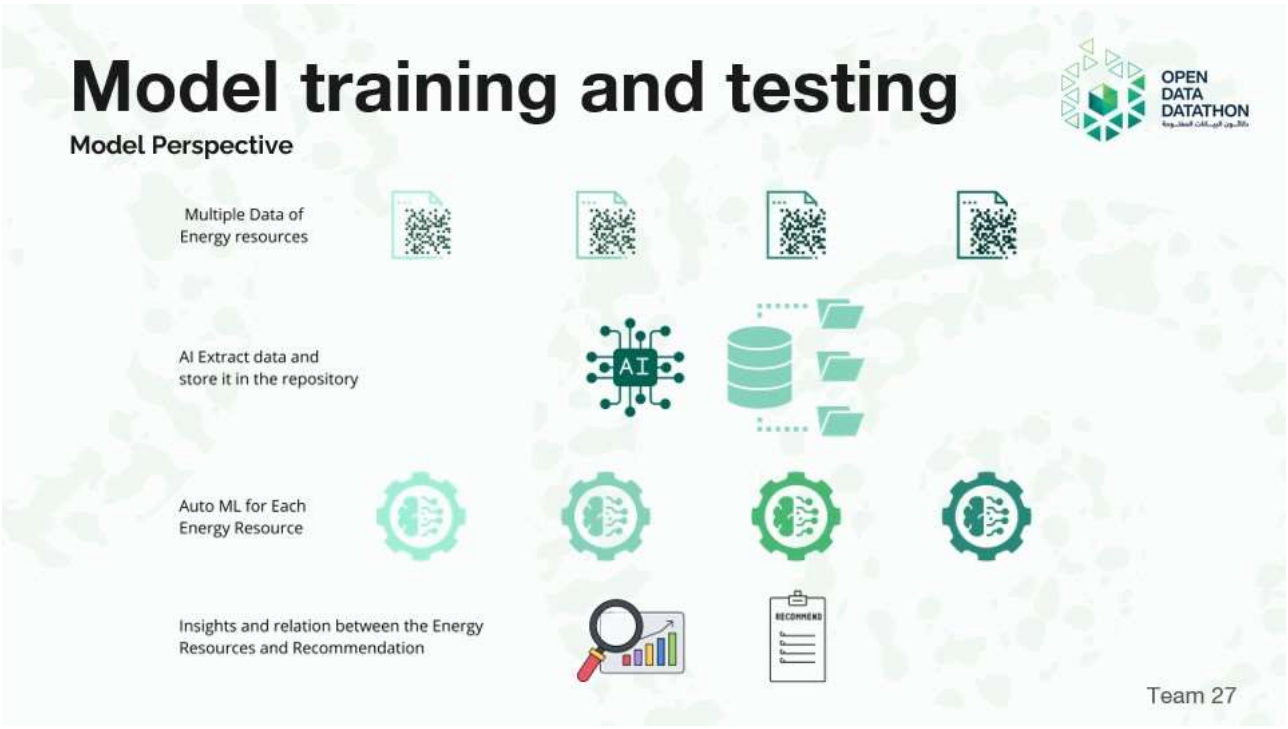

```
import seaborn as sns
import matplotlib.pyplot as plt

correlation_matrix = Electricity.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
<ipython-input-22-66e32487e3cd>:4: FutureWarning: The default value of numeric_only in D
  correlation_matrix = Electricity.corr()
```




```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

Electricity['Electricity Consumption'] = Electricity['Electricity Consumption'].round(0)

# Pivot the data
heatmap_data = Electricity.pivot_table(index='Region', columns='Year', values='Electricity Consumption', aggfunc='sum')

# Create the heatmap
plt.figure(figsize=(20, 4))
sns.heatmap(heatmap_data, cmap='YlGnBu', annot=True, fmt='.0f', linewidths=0.5)
plt.title('Electricity Consumption Heatmap')
plt.show()
```

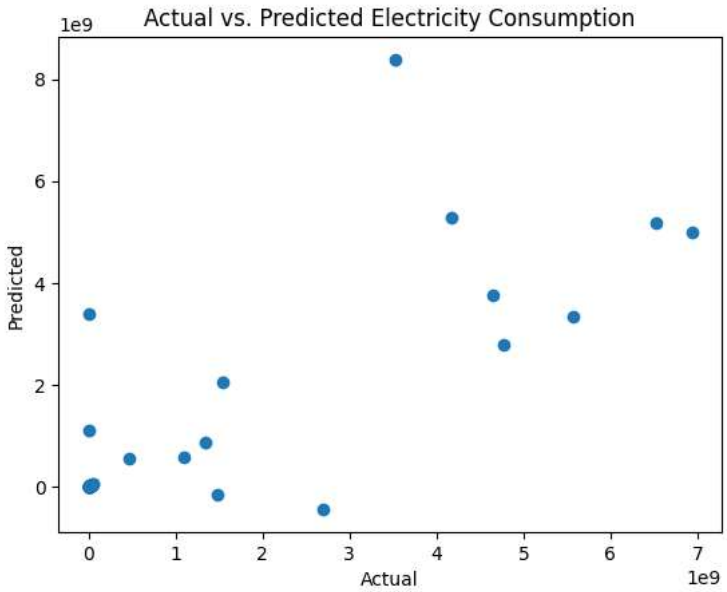





```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import re
import matplotlib.pyplot as plt


# Select features and target variable
X = Electricity[['Year', 'Region', 'Sector']]
y = Electricity['Electricity Consumption'].astype(str)

# Define a function to clean and convert to float
def clean_and_convert(value):
    try:
        cleaned_value = re.sub(r'[^\d.]', '', value)
        return float(cleaned_value)
    except ValueError:
        return np.nan

y = y.apply(clean_and_convert)

# Drop rows with NaN values in the target variable
X = X[~y.isna()]
y = y.dropna()

# Handle categorical variables (one-hot encoding)
X = pd.get_dummies(X, columns=['Region', 'Sector'], prefix=['region', 'sector'])

# Split the data 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)

# Standardize features (scaling)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
gb = GradientBoostingRegressor(random_state=42)
```

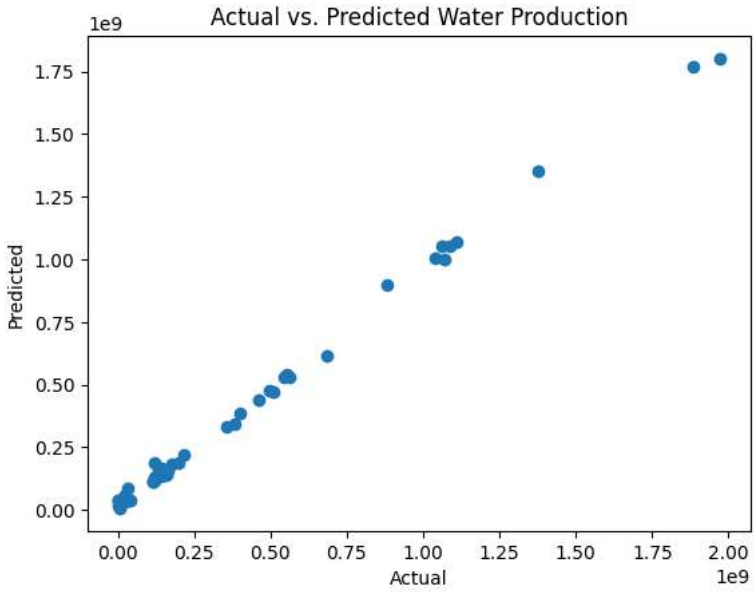
```
gb = GradientBoostingRegressor(random_state=42)
grid_search = GridSearchCV(estimator=gb, param_grid=param_grid, cv=5, n_jobs=-1, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_

# Train a Gradient Boosting Regressor with the best hyperparameters
model = GradientBoostingRegressor(random_state=42, **best_params)
model.fit(X_train, y_train)

# Cross-validation to evaluate the model
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_rmse = np.sqrt(-cv_scores)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Best Hyperparameters: {best_params}")
print(f"Cross-Validation RMSE: {cv_rmse.mean()}")
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
print(f"R-squared (R2): {r2}")

# Visualize actual vs. predicted values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs. Predicted Electricity Consumption")
plt.show()
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
Cross-Validation RMSE: 1428646137.842931
Mean Squared Error: 8.055366490126118e+17
Mean Absolute Error: 308503391.4692983
R-squared (R2): 0.6446403210517583
```



```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Select features and target variable
X_water = Water[['Date', 'Plant Name']]
y_water = Water['Value']

# Handle categorical variables (one-hot encoding or label encoding)
X_water = pd.get_dummies(X_water, columns=['Plant Name'], prefix='plant')

# Split the data into training and testing sets
X_train_water, X_test_water, y_train_water, y_test_water = train_test_split(X_water, y_water, test_size=0.2, random_state=42)

# Train a machine learning model (Random Forest Regressor)
model_water = RandomForestRegressor(n_estimators=100, random_state=42)
model_water.fit(X_train_water, y_train_water)

# Make predictions on the test set
y_pred_water = model_water.predict(X_test_water)

# Evaluate the model
mse_water = mean_squared_error(y_test_water, y_pred_water)
mae_water = mean_absolute_error(y_test_water, y_pred_water)
r2_water = r2_score(y_test_water, y_pred_water)

print(f"Mean Squared Error: {mse_water}")
print(f"Mean Absolute Error: {mae_water}")
print(f"R-squared (R2): {r2_water}")

# Visualize actual vs. predicted values
plt.scatter(y_test_water, y_pred_water)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs. Predicted Water Production")
plt.show()
```

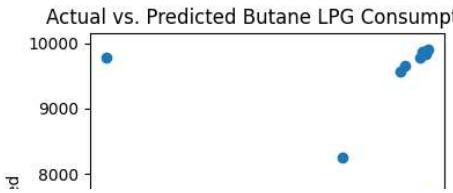
```
Mean Squared Error: 1554701014661073.2
Mean Absolute Error: 24652674.526900563
R-squared (R2): 0.9929981892100825
```

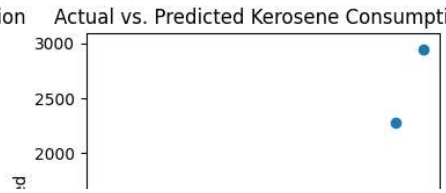


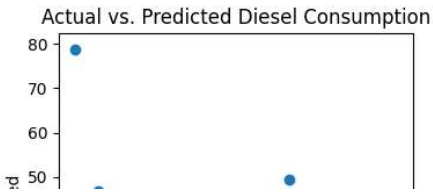
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Select features and target variables
X_gas = Gas[['Year', 'Region']]  # Add other relevant features
y_gas = Gas[['Butane LPG', 'Kerosene', 'Diesel']]

# Handle categorical variables (one-hot encoding)
X_gas = pd.get_dummies(X_gas, columns=['Region'], prefix='region')

# Split the data into training and testing sets
X_train_gas, X_test_gas, y_train_gas, y_test_gas = train_test_split(X_gas, y_gas, test_size=0.2, random_state=42)

# Train a multi-output machine learning model (Random Forest Regressor)
model_gas_multioutput = MultiOutputRegressor(RandomForestRegressor(n_estimators=100, random_state=42))
model_gas_multioutput.fit(X_train_gas, y_train_gas)

# Make predictions on the test set
y_pred_gas = model_gas_multioutput.predict(X_test_gas)

# Evaluate the model
mse_gas = mean_squared_error(y_test_gas, y_pred_gas)
mae_gas = mean_absolute_error(y_test_gas, y_pred_gas)
r2_gas = r2_score(y_test_gas, y_pred_gas)

print("For all three target variables:")
print(f"Mean Squared Error: {mse_gas}")
print(f"Mean Absolute Error: {mae_gas}")
print(f"R-squared (R2): {r2_gas}")

# Visualize actual vs. predicted values for all three target variables
plt.figure(figsize=(12, 4))
for i, gas_type in enumerate(['Butane LPG', 'Kerosene', 'Diesel']):
    plt.subplot(1, 3, i + 1)
    plt.scatter(y_test_gas[gas_type], y_pred_gas[:, i])
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual vs. Predicted {gas_type} Consumption")

plt.tight_layout()
plt.show()

# Calculate feature importances for 'Butane LPG'
feature_importances = model_gas_multioutput.estimators_[0].feature_importances_
feature_names = X_gas.columns
sorted_idx = np.argsort(feature_importances)[::-1]
top_features = 10  # You can change this number
plt.figure(figsize=(6, 4))
plt.title("Feature Importances for Butane LPG")
plt.bar(range(top_features), feature_importances[sorted_idx][:top_features], align="center")
plt.xticks(range(top_features), feature_names[sorted_idx][:top_features], rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Create a residual plot for 'Butane LPG'
residuals_gas = y_test_gas['Butane LPG'] - y_pred_gas[:, 0]
plt.figure(figsize=(4,3))
plt.scatter(y_pred_gas[:, 0], residuals_gas)
plt.title("Residual Plot for Butane LPG")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.axhline(y=0, color='r', linestyle='--')
plt.tight_layout()
plt.show()

# Calculate the distribution of errors for 'Butane LPG'
errors_gas = y_test_gas['Butane LPG'] - y_pred_gas[:, 0]
plt.figure(figsize=(4, 3))
plt.hist(errors_gas, bins=30, color='skyblue', edgecolor='black')
plt.title("Distribution of Errors for Butane LPG")
plt.xlabel("Errors")
plt.ylabel("Frequency")
```

```
plt.tight_layout()
plt.show()

# Cross-validation to evaluate the model for 'Butane LPG'
cv_scores_gas = cross_val_score(model_gas_multioutput.estimators_[0], X_train_gas, y_train_gas['Butane LPG'], cv=5, scoring='neg_mean_square
cv_rmse_gas = np.sqrt(-cv_scores_gas)
print(f"Cross-Validation RMSE for Butane LPG: {cv_rmse_gas.mean()}")

# Similar analyses for 'Kerosene' and 'Diesel'
```

```
For all three target variables:
Mean Squared Error: 3294676.0161636365
Mean Absolute Error: 603.770303030303
R-squared (R2): 0.05607217306790716
```







```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Select features and target variable
X_sun = Sun[['Year', 'Region']]  # Add other relevant features
y_sun = Sun['sun energy rate']

# Handle categorical variables (one-hot encoding)
X_sun = pd.get_dummies(X_sun, columns=['Region'], prefix='region')

# Split the data into training and testing sets
X_train_sun, X_test_sun, y_train_sun, y_test_sun = train_test_split(X_sun, y_sun, test_size=0.2, random_state=42)

# Train a machine learning model (Gradient Boosting Regressor)
model_sun = GradientBoostingRegressor(n_estimators=100, random_state=42)
model_sun.fit(X_train_sun, y_train_sun)

# Make predictions on the test set
y_pred_sun = model_sun.predict(X_test_sun)

# Evaluate the model
mse_sun = mean_squared_error(y_test_sun, y_pred_sun)
mae_sun = mean_absolute_error(y_test_sun, y_pred_sun)
r2_sun = r2_score(y_test_sun, y_pred_sun)

print(f"Mean Squared Error: {mse_sun}")
print(f"Mean Absolute Error: {mae_sun}")
print(f"R-squared (R2): {r2_sun}")

# Visualize actual vs. predicted values
plt.scatter(y_test_sun, y_pred_sun)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs. Predicted Sun Energy Rate")
plt.show()

# Calculate feature importances
feature_importances = model_sun.feature_importances_
feature_names = X_sun.columns
sorted_idx = np.argsort(feature_importances)[::-1]
top_features = 10  # You can change this number
plt.figure(figsize=(6, 4))
plt.title("Feature Importances")
plt.bar(range(top_features), feature_importances[sorted_idx][:top_features], align="center")
plt.xticks(range(top_features), feature_names[sorted_idx][:top_features], rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Create a residual plot
residuals = y_test_sun - y_pred_sun
plt.figure(figsize=(4, 3))
plt.scatter(y_pred_sun, residuals)
plt.title("Residual Plot")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.axhline(y=0, color='r', linestyle='--')
plt.tight_layout()
plt.show()

# Calculate the distribution of errors
errors = y_test_sun - y_pred_sun
plt.figure(figsize=(4, 3))
plt.hist(errors, bins=30, color='skyblue', edgecolor='black')
plt.title("Distribution of Errors")
plt.xlabel("Errors")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# Cross-validation to evaluate the model
cv_scores = cross_val_score(model_sun, X_train_sun, y_train_sun, cv=5, scoring='neg_mean_squared_error')
cv_rmse = np.sqrt(-cv_scores)
```

```
print(f"Cross-Validation RMSE: {cv_rmse.mean()}")
```