In [47]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from scipy.optimize import curve_fit
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [81]:
```
data = pd.read_csv('final.csv')
data.tail()
```

Out[81]:

	Entity	Year	Other Renewables	Hydroelectric	Oil	Gas	Coal
185	United Kingdom	2018	0.01	5.44	9.34	131.49	16.83
186	United Kingdom	2019	0.01	5.93	9.20	131.99	6.92
187	United Kingdom	2020	0.00	6.86	10.33	111.42	5.49
188	United Kingdom	2021	0.00	5.50	10.89	123.17	6.51
189	United Kingdom	2022	0.00	5.32	12.55	125.30	5.57

In [82]:
```
data.rename(columns={'Entity': 'Country'}, inplace=True)
```

In [83]:
```
data = data.drop(['Other Renewables', 'Hydroelectric'], axis=1)
```

In [84]:
```
data.to_csv('final_1.csv', index=False, header=True)
```

In [85]:
```
df = pd.read_csv('final_1.csv')
df
```

Out[85]:

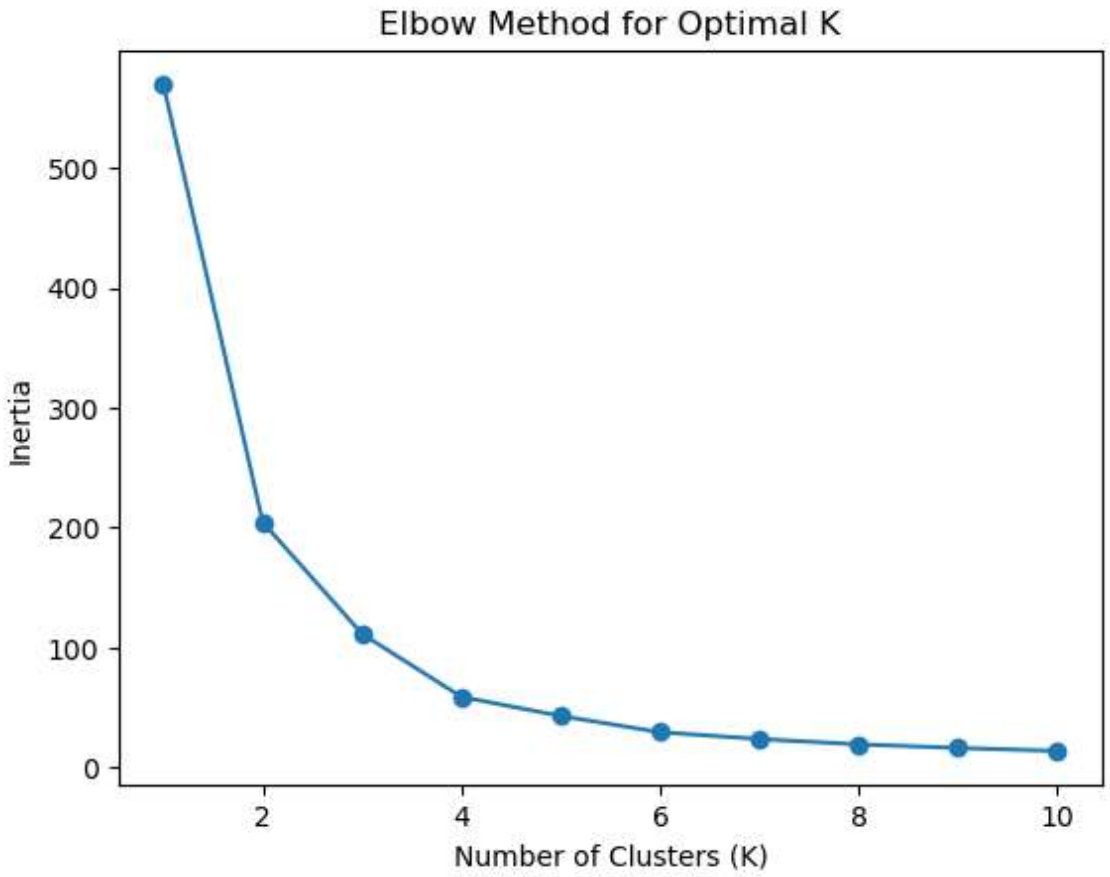
	Country	Year	Oil	Gas	Coal
0	Asia	1985	464.66388	345.53317	654.89886
1	Asia	1986	467.65207	369.96634	724.53830
2	Asia	1987	485.15662	397.68027	804.49884
3	Asia	1988	530.07983	423.47736	860.65796
4	Asia	1989	562.54740	467.87780	936.60070
...	...	...	...	...	...
185	United Kingdom	2018	9.34000	131.49000	16.83000
186	United Kingdom	2019	9.20000	131.99000	6.92000
187	United Kingdom	2020	10.33000	111.42000	5.49000
188	United Kingdom	2021	10.89000	123.17000	6.51000
189	United Kingdom	2022	12.55000	125.30000	5.57000

190 rows × 5 columns

Elbow Method

In [87]:

```
features_for_clustering = df[['Oil', 'Gas', 'Coal']]

# Normalize the data
scaler = StandardScaler()
normalized_features = scaler.fit_transform(features_for_clustering)

import warnings

# Suppress KMeans memory leak warning on Windows
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn.cluste

# Elbow Method to find Optimal k
inertia = []

# Perform the Elbow Method for different values of K
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(normalized_features)
    inertia.append(kmeans.inertia_)

# Plotting the Elbow Method
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.show()

```



Applying k-means clustering

In [88]:
```
k = 3

kmeans = KMeans(n_clusters=k, random_state=42)
df['Cluster'] = kmeans.fit_predict(normalized_features)
```

In [89]:
```
df
```

Out[89]:

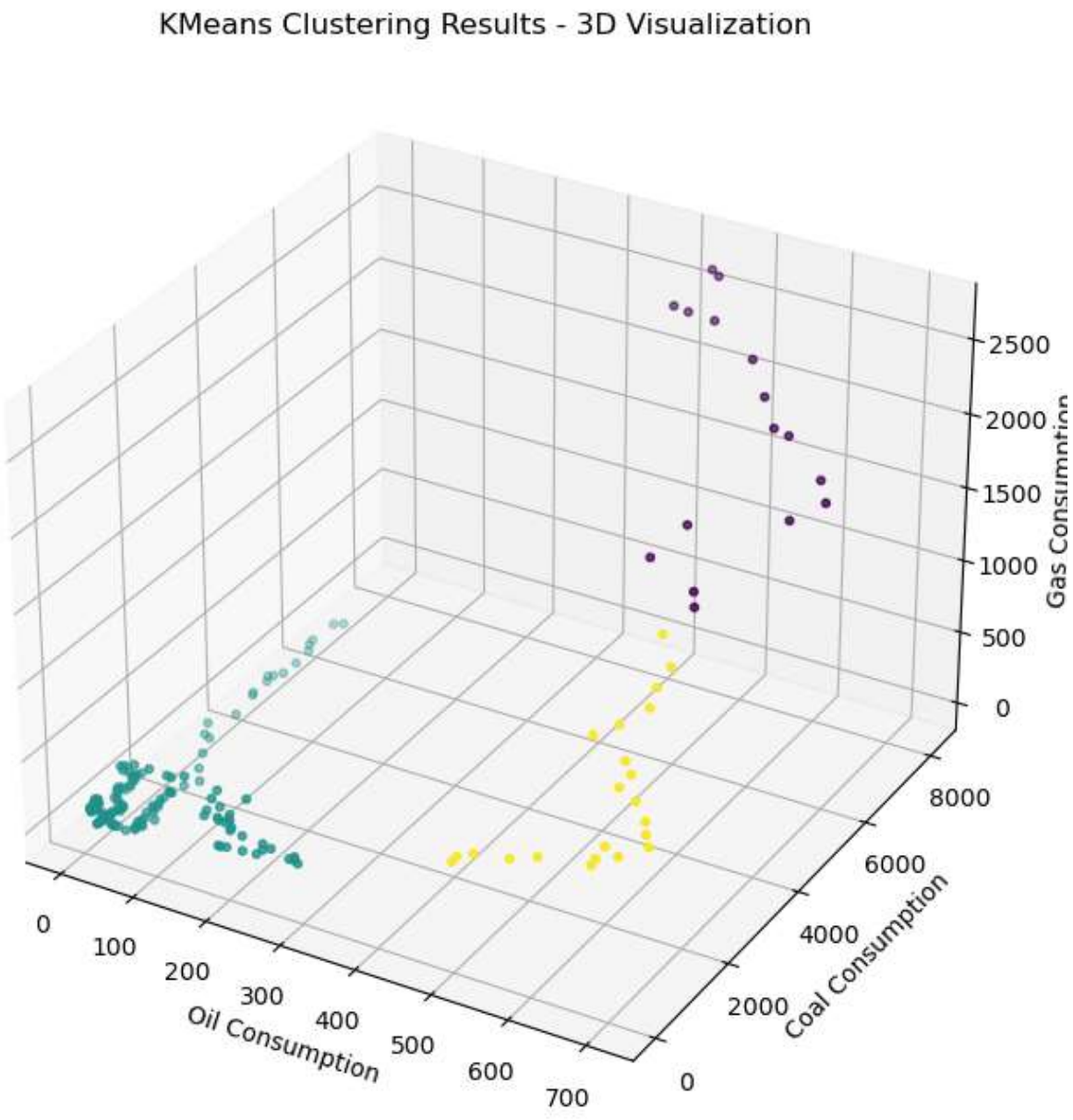
	Country	Year	Oil	Gas	Coal	Cluster
0	Asia	1985	464.66388	345.53317	654.89886	2
1	Asia	1986	467.65207	369.96634	724.53830	2
2	Asia	1987	485.15662	397.68027	804.49884	2
3	Asia	1988	530.07983	423.47736	860.65796	2
4	Asia	1989	562.54740	467.87780	936.60070	2
...	...	...	...	...	...	...
185	United Kingdom	2018	9.34000	131.49000	16.83000	1
186	United Kingdom	2019	9.20000	131.99000	6.92000	1
187	United Kingdom	2020	10.33000	111.42000	5.49000	1
188	United Kingdom	2021	10.89000	123.17000	6.51000	1
189	United Kingdom	2022	12.55000	125.30000	5.57000	1

190 rows × 6 columns

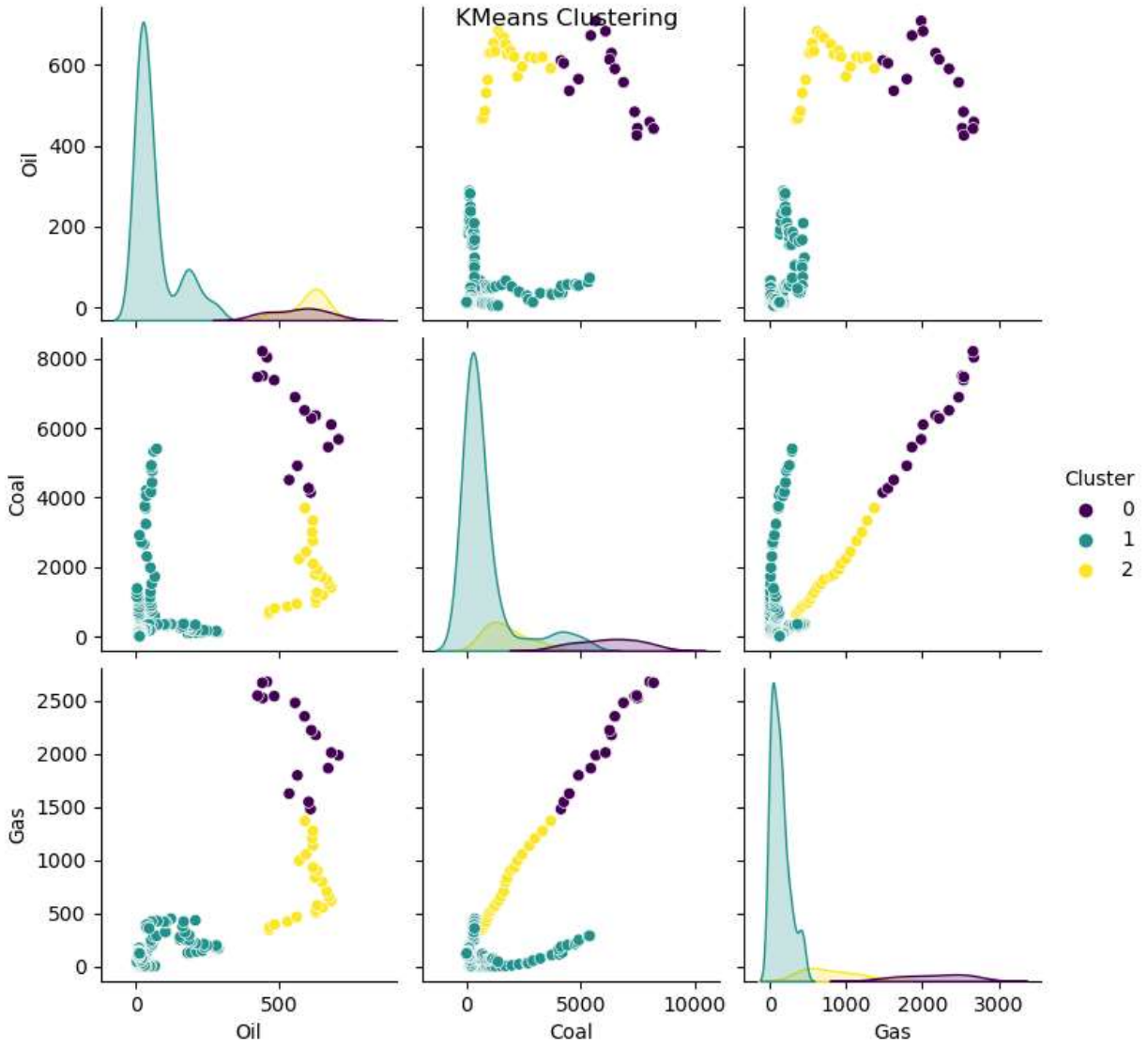
Visualization

In [90]:

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Assuming df contains your DataFrame with 'oil', 'coal', 'gas', 'hydroelectr
ax.scatter(df['Oil'], df['Coal'], df['Gas'], c=df['Cluster'], cmap='viridis',
ax.set_xlabel('Oil Consumption')
ax.set_ylabel('Coal Consumption')
ax.set_zlabel('Gas Consumption')
ax.set_title('KMeans Clustering Results - 3D Visualization')
plt.show()

```



In [92]:
```
import seaborn as sns

sns.pairplot(df, hue='Cluster', vars=['Oil', 'Coal', 'Gas'], palette='viridis
plt.suptitle('KMeans Clustering')
plt.show()

```



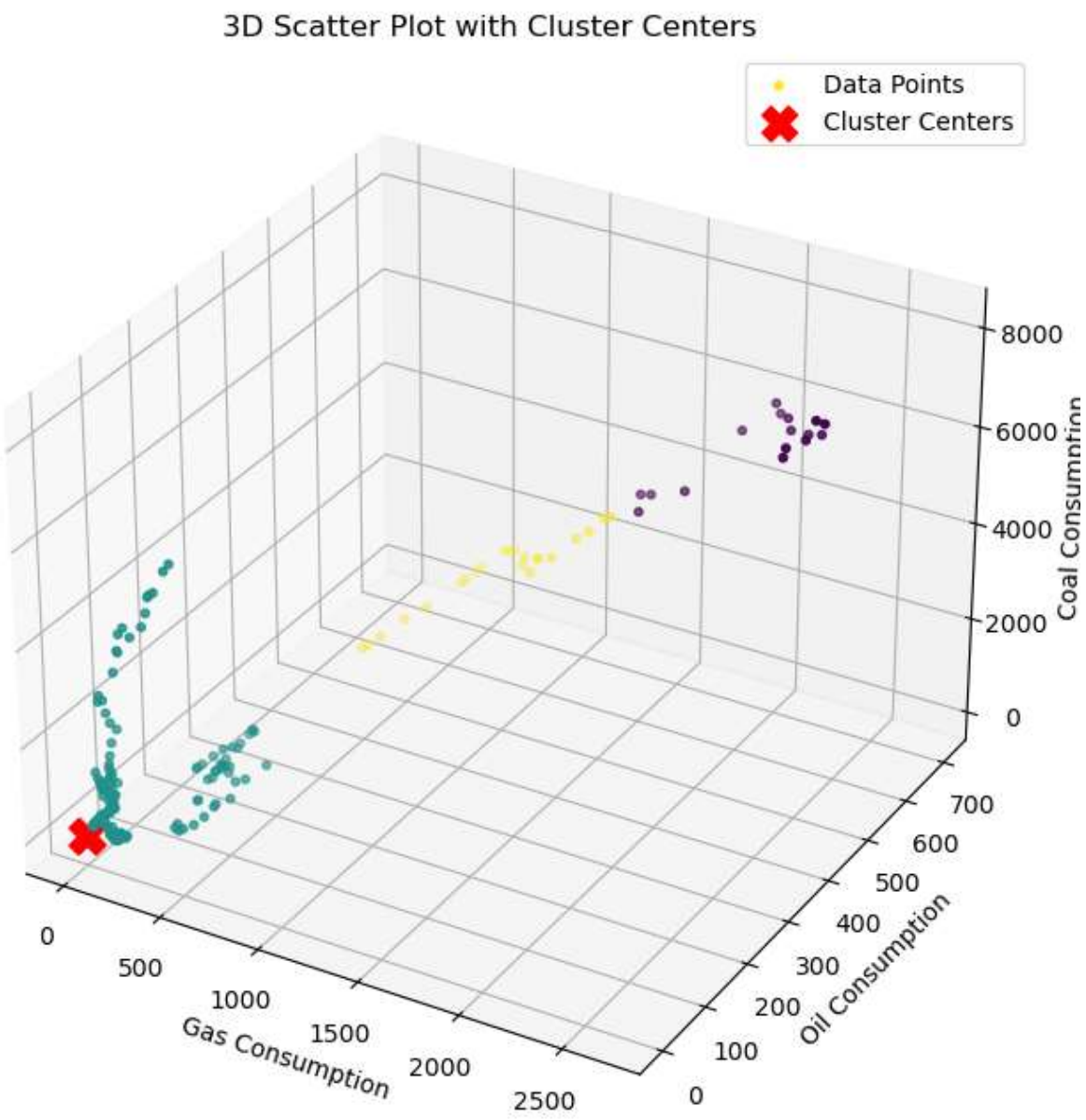
Analyze Cluster Centers

In [93]:
```
cluster_centers = kmeans.cluster_centers_
print("Cluster Centers:")
print(cluster_centers)
```

```
Cluster Centers:
[[ 1.79062913  2.94075205  2.53772761]
 [-0.4720725  -0.40183665 -0.29492772]
 [ 1.95931607  0.63759722  0.19206234]]
```

In [95]:

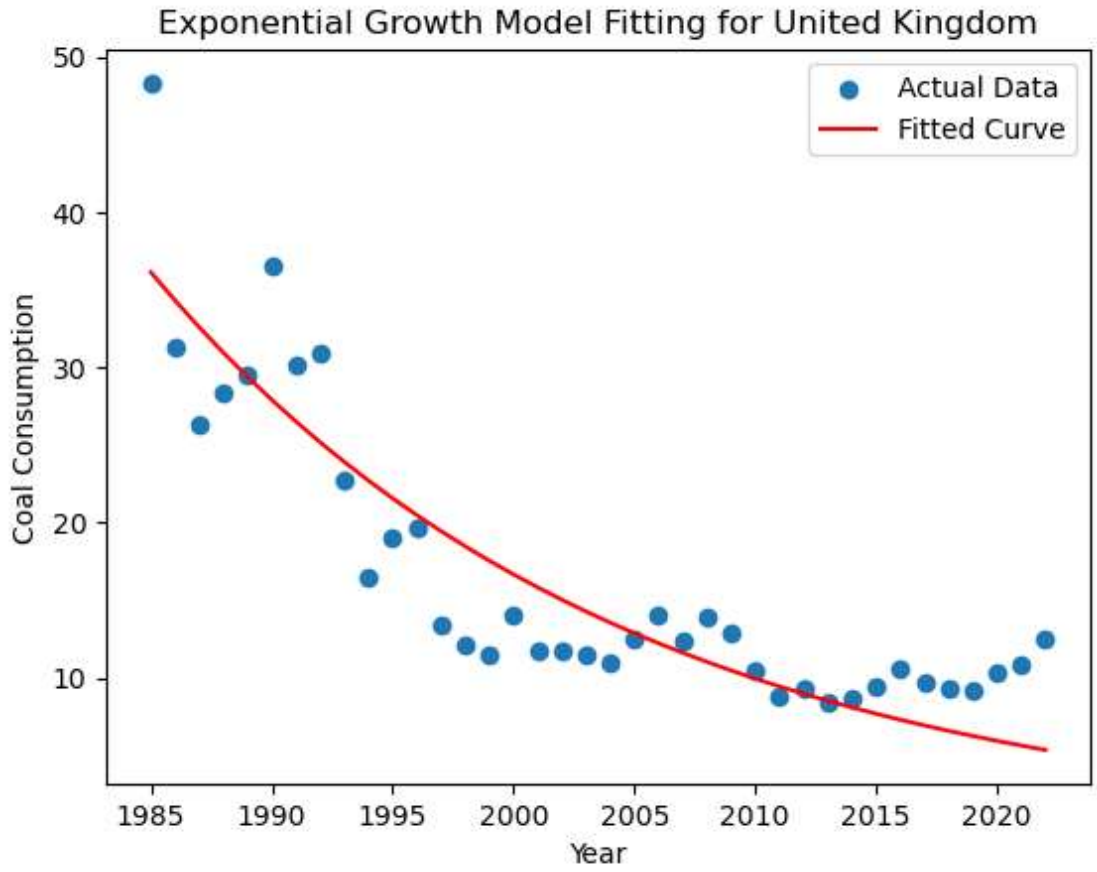
```
# Visualization: 3D Scatter Plot
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plotting the actual data points
ax.scatter(df['Gas'], df['Oil'], df['Coal'], c=df['Cluster'], cmap='viridis',

# Plotting the cluster centers
cluster_centers = kmeans.cluster_centers_
ax.scatter(cluster_centers[:, 0], cluster_centers[:, 1], cluster_centers[:, 2

# Customize the plot
ax.set_xlabel('Gas Consumption')
ax.set_ylabel('Oil Consumption')
ax.set_zlabel('Coal Consumption')
ax.set_title('3D Scatter Plot with Cluster Centers')
ax.legend()

# Show the plot
plt.show()

```



Model Fitting

In [96]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Assuming df contains your DataFrame with 'Year', 'Hydroelectric', and 'Clust
selected_country = 'United Kingdom'  # Replace with the country you are intere
country_data = df[df['Country'] == selected_country]

# Extract the years and Hydroelectric values
years = country_data['Year']
hydro_values = country_data['Oil']

# Define the exponential growth model
def exponential_growth(t, a, b):
    return a * np.exp(b * (t - years.min()))

# Provide an initial guess for parameters 'a' and 'b'
initial_guess = [1.0, 0.01]

# Fit the model to the data
params, covariance = curve_fit(exponential_growth, years, hydro_values, p0=ini

# Predict using the fitted parameters
predicted_values = exponential_growth(years, *params)

# Plot the original data and the fitted curve
plt.scatter(years, hydro_values, label='Actual Data')
plt.plot(years, predicted_values, color='red', label='Fitted Curve')
plt.xlabel('Year')
plt.ylabel('Coal Consumption')
plt.title(f'Exponential Growth Model Fitting for {selected_country}')
plt.legend()
plt.show()

# Print the fitted parameters
print("Fitted Parameters (a, b):", params)

# Calculate confidence intervals
err_ranges = np.sqrt(np.diag(covariance))
lower_bound = params - 1.96 * err_ranges
upper_bound = params + 1.96 * err_ranges

# Print confidence intervals
print("Confidence Intervals (95%):")
print("Lower Bound:", lower_bound)
print("Upper Bound:", upper_bound)

```




```
Fitted Parameters (a, b): [36.12350383 -0.05160788]
Confidence Intervals (95%):
Lower Bound: [32.18074074 -0.06099872]
Upper Bound: [40.06626692 -0.04221704]
```

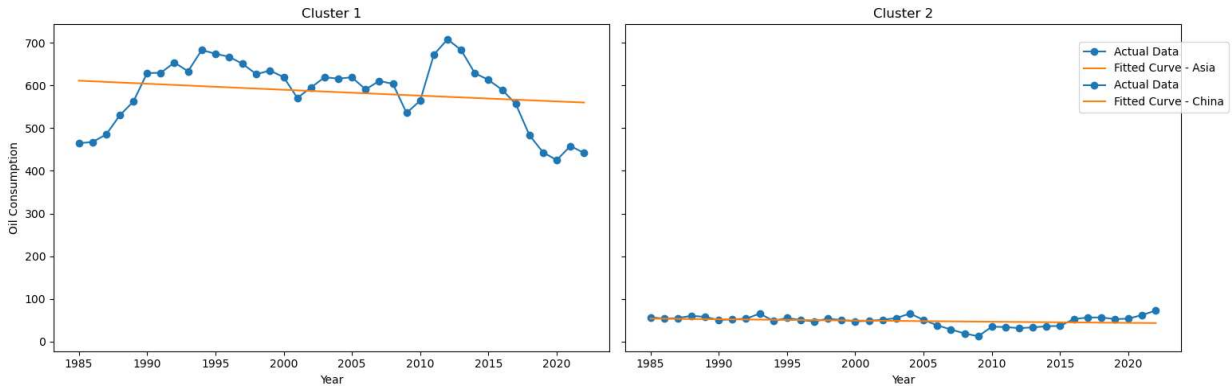
Comparative Analysis

In [98]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.optimize import curve_fit

# Read the data
df = pd.read_csv('final_1.csv')

# Select features for clustering
features_for_clustering = ['Oil', 'Gas', 'Coal']

# Perform KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(df[features_for_clustering])

# Create a DataFrame to store selected countries from each cluster
selected_countries = pd.DataFrame(columns=df.columns)

# Set to keep track of selected countries
selected_set = set()

# Choose one country from each cluster for further analysis
for cluster in df['Cluster'].unique():
    # Check if there are available countries in the cluster
    available_countries = df[(df['Cluster'] == cluster) & (~df['Country'].isi
    if not available_countries.empty:
        # Select the first country not already in the set
        selected_country = available_countries.iloc[0]

        # Add the selected country to the set
        selected_set.add(selected_country['Country'])

        # Append the selected country to the DataFrame
        selected_countries = selected_countries.append(selected_country)

# Function to fit an exponential growth model
def exponential_growth(t, a, b, t_min):
    return a * np.exp(b * (t - t_min))

# Create subplots for each cluster
fig, axes = plt.subplots(nrows=1, ncols=len(selected_countries), figsize=(15,

# Compare trends for each cluster
for ax, (index, country_row) in zip(axes, selected_countries.iterrows()):
    selected_country = country_row['Country']
    cluster = country_row['Cluster']

    # Extract data for the selected country
    country_data = df[df['Country'] == selected_country]

    # Extract the years and Oil values
    years = country_data['Year']
    oil_values = country_data['Oil']

    # Fit the exponential growth model
    params, _ = curve_fit(exponential_growth, years, oil_values, p0=[1.0, 0.0
```

```
58
59      # Predict using the fitted parameters
60      predicted_values = exponential_growth(years, *params)
61
62      # Plot the original data and the fitted curve
63      ax.plot(years, oil_values, 'o-', label='Actual Data')
64      ax.plot(years, predicted_values, label=f'Fitted Curve - {selected_country}')
65
66      # Customize each subplot
67      ax.set_xlabel('Year')
68      ax.set_title(f'Cluster {cluster}')
69
70  # Customize the common y-axis label
71  axes[0].set_ylabel('Oil Consumption')
72
73  # Add a legend to the entire figure
74  fig.legend(loc='upper left', bbox_to_anchor=(0.9, 0.9))
75
76  # Adjust layout for better spacing
77  plt.tight_layout()
78  plt.show()
79
```

```
C:\Users\SSC\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: RuntimeWar
ning: overflow encountered in exp
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



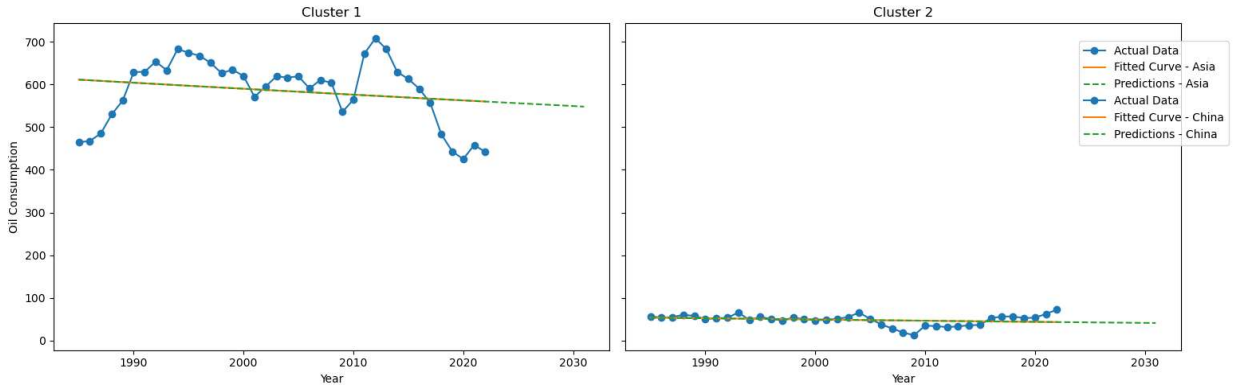
Prediction For Future Years

In [103]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.optimize import curve_fit

# Read the data
df = pd.read_csv('final_1.csv')

# Select features for clustering
features_for_clustering = ['Oil', 'Gas', 'Coal']

# Perform KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(df[features_for_clustering])

# Create a DataFrame to store selected countries from each cluster
selected_countries = pd.DataFrame(columns=df.columns)

# Set to keep track of selected countries
selected_set = set()

# Choose one country from each cluster for further analysis
for cluster in df['Cluster'].unique():
    # Check if there are available countries in the cluster
    available_countries = df[(df['Cluster'] == cluster) & (~df['Country'].isi
    if not available_countries.empty:
        # Select the first country not already in the set
        selected_country = available_countries.iloc[0]

        # Add the selected country to the set
        selected_set.add(selected_country['Country'])

        # Append the selected country to the DataFrame
        selected_countries = selected_countries.append(selected_country)

# Function to fit an exponential growth model
def exponential_growth(t, a, b, t_min):
    return a * np.exp(b * (t - t_min))

# Create subplots for each cluster
fig, axes = plt.subplots(nrows=1, ncols=len(selected_countries), figsize=(15,

# Compare trends for each cluster and make predictions
for ax, (index, country_row) in zip(axes, selected_countries.iterrows()):
    selected_country = country_row['Country']
    cluster = country_row['Cluster']

    # Extract data for the selected country
    country_data = df[df['Country'] == selected_country]

    # Extract the years and Hydroelectric values
    years = country_data['Year']
    oil_values = country_data['Oil']

    # Fit the exponential growth model
    params, _ = curve_fit(exponential_growth, years, oil_values, p0=[1.0, 0.0
```

```
58
59      # Predict using the fitted parameters for the next 10 years
60      future_years = np.arange(years.min(), years.max() + 10)
61      future_predictions = exponential_growth(future_years, *params)
62
63      # Plot the original data, the fitted curve, and the predictions
64      ax.plot(years, oil_values, 'o-', label='Actual Data')
65      ax.plot(years, exponential_growth(years, *params), label=f'Fitted Curve -
66      ax.plot(future_years, future_predictions, label=f'Predictions - {selected_
67
68      # Customize each subplot
69      ax.set_xlabel('Year')
70      ax.set_title(f'Cluster {cluster}')
71
72  # Customize the common y-axis label
73  axes[0].set_ylabel('Oil Consumption')
74
75  # Add a legend to the entire figure
76  fig.legend(loc='upper left', bbox_to_anchor=(0.9, 0.9))
77
78  # Adjust layout for better spacing
79  plt.tight_layout()
80  plt.show()
81
```

```
C:\Users\SSC\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: RuntimeWar
ning: overflow encountered in exp
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



In [ ]:
```
1
```