Importing Libraries

In [1]:

```
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

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

Module Error

In [2]:

```
"""
Module errors. Contains:
error_prop Calculates the error range caused by the uncertainty of the fit
    parameters. Covariances are taken into account.
cover_to_corr: Converts covariance matrix into correlation matrix.
"""


import numpy as np


def error_prop(x, func, parameter, covar):
    """
    Calculates 1 sigma error ranges for number or array. It uses error
    propagation with variances and covariances taken from the covar matrix.
    Derivatives are calculated numerically.

    """

    # initiate sigma the same shape as parameter

    var = np.zeros_like(x)   # initialise variance vektor
    # Nested loop over all combinations of the parameters
    for i in range(len(parameter)):
        # derivative with respect to the ith parameter
        deriv1 = deriv(x, func, parameter, i)

        for j in range(len(parameter)):
            # derivative with respct to the jth parameter
            deriv2 = deriv(x, func, parameter, j)



            # multiplied with the i-jth covariance
            # variance vektor
            var = var + deriv1*deriv2*covar[i, j]

    # Check for division by zero or invalid values
    mask = np.isinf(var) | np.isnan(var)
    var[mask] = 0  # Set invalid values to 0 or handle them appropriately

    sigma = np.sqrt(var)
    return sigma


def deriv(x, func, parameter, ip):
    """
    Calculates numerical derivatives from function
    values at parameter +/- delta.  Parameter is the vector with parameter
    values. ip is the index of the parameter to derive the derivative.

    """

    # print("in", ip, parameter[ip])
    # create vector with zeros and insert delta value for relevant parameter
    # delta is calculated as a small fraction of the parameter value
    scale = 1e-6   # scale factor to calculate the derivative
    delta = np.zeros_like(parameter, dtype=float)
    val = scale * np.abs(parameter[ip])
    delta[ip] = val  #scale * np.abs(parameter[ip])

    diff = 0.5 * (func(x, *parameter+delta) - func(x, *parameter-delta))
    dfdx = diff / val

    return dfdx


def covar_to_corr(covar):
    """ Converts the covariance matrix into a correlation matrix """

    # extract variances from the diagonal and calculate std. dev.
    sigma = np.sqrt(np.diag(covar))
```

```
    # construct matrix containing the sigma values
    matrix = np.outer(sigma, sigma)
    # and divide by it
    corr = covar/matrix

    return corr


```

Tools for Support clusters

In [3]:

```
""" Tools to support clustering: correlation heatmap, normaliser and scale
(cluster centres) back to original scale, check for mismatching entries """


# def map_corr(df, size=6):
#     """Function creates heatmap of correlation matrix for each pair of
#     columns in the dataframe.

#     Input:
#         df: pandas DataFrame
#         size: vertical and horizontal size of the plot (in inch)

#     The function does not have a plt.show() at the end so that the user
#     can savethe figure.
#     """

#     import matplotlib.pyplot as plt  # ensure pyplot imported

#     corr = df.corr()
#     plt.figure(figsize=(size, size))
#     # fig, ax = plt.subplots()
#     plt.matshow(corr, cmap='coolwarm', location="bottom")
#     # setting ticks to column names
#     plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
#     plt.yticks(range(len(corr.columns)), corr.columns)

#     plt.colorbar()
#     # no plt.show() at the end

def map_corr(df, size=15):
    """Function creates a heatmap of the correlation matrix for each pair of
    columns in the dataframe.

    Input:
        df: pandas DataFrame
        size: vertical and horizontal size of the plot (in inches)

    The function does not have a plt.show() at the end so that the user
    can save the figure.
    """
    corr = df.corr(numeric_only=True)

    plt.figure(figsize=(size, size))

    cmap = sns.color_palette("coolwarm", as_cmap=True)

    # Plot the heatmap without a mask
    sns.heatmap(corr, cmap=cmap, annot=True, fmt=".2f", linewidths=.5)

    plt.show()

def scaler(df):
    """ Expects a dataframe and normalises all
        columnsto the 0-1 range. It also returns
        dataframes with minimum and maximum for
        transforming the cluster centres"""

    # Uses the pandas methods
    df_min = df.min()
    df_max = df.max()

    df = (df-df_min) / (df_max - df_min)

    return df, df_min, df_max


def backscale(arr, df_min, df_max):
    """ Expects an array of normalized cluster centres and scales
        it back. Returns numpy array.  """

    # Convert df_min and df_max to numpy arrays for indexing
    minima = df_min.to_numpy()
```

```
    maxima = df_max.to_numpy()

    # loop over the "columns" of the numpy array
    for i in range(arr.shape[1]):
        arr[:, i] = arr[:, i] * (maxima[i] - minima[i]) + minima[i]

    return arr


def get_diff_entries(df1, df2, column):
    """ Compares the values of column in df1 and the column with the same
    name in df2. A list of mismatching entries is returned. The list will be
    empty if all entries match. """

    import pandas as pd  # to be sure

    # merge dataframes keeping all rows
    df_out = pd.merge(df1, df2, on=column, how="outer")
    print("total entries", len(df_out))
    # merge keeping only rows in common
    df_in = pd.merge(df1, df2, on=column, how="inner")
    print("entries in common", len(df_in))
    df_in["exists"] = "Y"

    # merge again
    df_merge = pd.merge(df_out, df_in, on=column, how="outer")

    # extract columns without "Y" in exists
    df_diff = df_merge[(df_merge["exists"] != "Y")]
    diff_list = df_diff[column].to_list()

    return diff_list

```

Loading Dataset

In [4]:
```
df = pd.read_csv('co2.csv')
df
```

Out[4]:

	Country Name	Country Code	Indicator Name	Indicator Code	1990	1991	1992	1993	1994	1995	...	2013	
0	Africa Eastern and Southern	AFE	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	0.982975	0.942212	0.907936	0.909550	0.913413	0.933001	...	1.001154	1.
1	Afghanistan	AFG	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	0.191389	0.180674	0.126517	0.109106	0.096638	0.088781	...	0.298088	0.
2	Africa Western and Central	AFW	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	0.470111	0.521084	0.558013	0.513859	0.462384	0.492656	...	0.481623	0.
3	Angola	AGO	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	0.554941	0.545807	0.544413	0.710961	0.839266	0.914265	...	1.031044	1.
4	Albania	ALB	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	1.844035	1.261054	0.689644	0.644008	0.649938	0.612055	...	1.656390	1.
...	...	...	...	...	...	...	...	...	...	...	...	...	
232	Samoa	WSM	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	0.529176	0.579131	0.606011	0.656505	0.597318	0.666659	...	0.983800	1.
233	Yemen, Rep.	YEM	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	0.496616	0.611585	0.632544	0.570608	0.600495	0.654007	...	1.031167	0.
234	South Africa	ZAF	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	6.209373	5.922276	5.717823	5.795258	5.826213	6.007616	...	8.116435	8
235	Zambia	ZMB	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	0.356578	0.364978	0.352722	0.304005	0.252979	0.245217	...	0.278215	0.
236	Zimbabwe	ZWE	CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC	1.634929	1.763473	1.735620	1.581818	1.469850	1.408363	...	0.901248	0.

237 rows × 37 columns

Cleaning & Preprocessing

In [5]:
```
df = df.dropna(axis=1)
df.shape
```

Out[5]: (237, 34)

In [6]:
```
df = df.drop(['Country Code', 'Indicator Code'], axis=1)
```

In [7]:
```
df.head()
```

Out[7]:

	Country Name	Indicator Name	1991	1992	1993	1994	1995	1996	1997	1998	...	2011	2012	
0	Africa Eastern and Southern	CO2 emissions (metric tons per capita)	0.942212	0.907936	0.909550	0.913413	0.933001	0.943200	0.962203	0.963157	...	0.976840	0.989585	1
1	Afghanistan	CO2 emissions (metric tons per capita)	0.180674	0.126517	0.109106	0.096638	0.088781	0.082267	0.075559	0.071270	...	0.408965	0.335061	0
2	Africa Western and Central	CO2 emissions (metric tons per capita)	0.521084	0.558013	0.513859	0.462384	0.492656	0.554305	0.540062	0.506709	...	0.451578	0.452101	0
3	Angola	CO2 emissions (metric tons per capita)	0.545807	0.544413	0.710961	0.839266	0.914265	1.073630	1.086325	1.091173	...	0.983787	0.947583	1
4	Albania	CO2 emissions (metric tons per capita)	1.261054	0.689644	0.644008	0.649938	0.612055	0.621206	0.469831	0.576804	...	1.768109	1.565921	1

5 rows × 32 columns

In [34]:
```
df.to_csv('final.csv', index=False)
```

In [8]:
```
df1 = df.drop(['Country Name','Indicator Name'], axis=1)
df1.head()
```

Out[8]:

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2011	2012	
0	0.942212	0.907936	0.909550	0.913413	0.933001	0.943200	0.962203	0.963157	0.902134	0.891352	...	0.976840	0.989585	1.00
1	0.180674	0.126517	0.109106	0.096638	0.088781	0.082267	0.075559	0.071270	0.058247	0.055167	...	0.408965	0.335061	0.29
2	0.521084	0.558013	0.513859	0.462384	0.492656	0.554305	0.540062	0.506709	0.502905	0.521689	...	0.451578	0.452101	0.48
3	0.545807	0.544413	0.710961	0.839266	0.914265	1.073630	1.086325	1.091173	1.109791	0.988416	...	0.983787	0.947583	1.03
4	1.261054	0.689644	0.644008	0.649938	0.612055	0.621206	0.469831	0.576804	0.960297	1.031568	...	1.768109	1.565921	1.65

5 rows × 30 columns

In [9]:
```
df1.columns
```

Out[9]:
```
Index(['1991', '1992', '1993', '1994', '1995', '1996', '1997', '1998', '1999',
       '2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008',
       '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017',
       '2018', '2019', '2020'],
      dtype='object')
```

In [10]:
```
scaled_df, df_min, df_max = scaler(df1)
```

In [11]:
```
scaled_df
```

Out[11]:

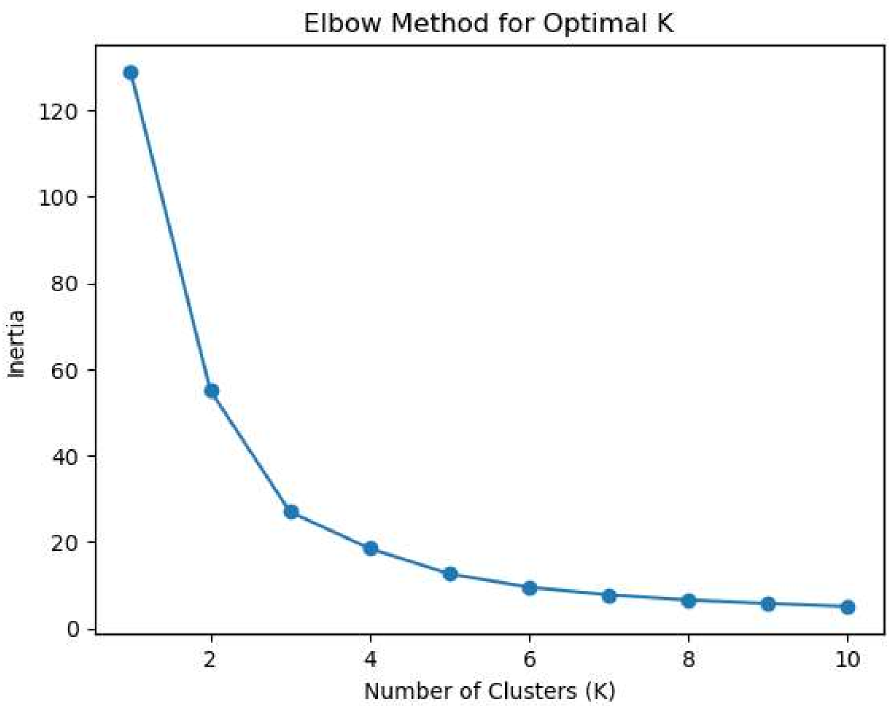
	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2011	2012	
0	0.028702	0.029154	0.026608	0.024733	0.025232	0.023837	0.020864	0.021114	0.019077	0.020085	...	0.024744	0.024084	0.
1	0.005504	0.004062	0.003192	0.002617	0.002401	0.002079	0.001638	0.001562	0.001232	0.001243	...	0.009777	0.007533	0.
2	0.015874	0.017918	0.015032	0.012520	0.013324	0.014009	0.011710	0.011108	0.010635	0.011755	...	0.010900	0.010492	0.
3	0.016627	0.017481	0.020798	0.022725	0.024726	0.027134	0.023555	0.023921	0.023468	0.022272	...	0.024928	0.023022	0.
4	0.038415	0.022144	0.018840	0.017598	0.016553	0.015700	0.010187	0.012645	0.020307	0.023244	...	0.045599	0.038658	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
232	0.017642	0.019459	0.019205	0.016174	0.018029	0.018807	0.015679	0.017756	0.016840	0.018197	...	0.025308	0.024344	0.
233	0.018631	0.020311	0.016692	0.016260	0.017687	0.016427	0.014753	0.015507	0.016412	0.018181	...	0.022742	0.019323	0.
234	0.180409	0.183597	0.169534	0.157757	0.162472	0.155074	0.138179	0.141613	0.126917	0.136923	...	0.204790	0.202237	0.
235	0.011118	0.011326	0.008893	0.006850	0.006632	0.005002	0.005426	0.005083	0.003877	0.004117	...	0.004635	0.005972	0.
236	0.053720	0.055730	0.046274	0.039799	0.038088	0.033602	0.026385	0.026715	0.028550	0.025854	...	0.021979	0.021849	0.

237 rows × 30 columns

Elbow Method

In [12]:

```
import warnings

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

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(scaled_df)
    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()
```

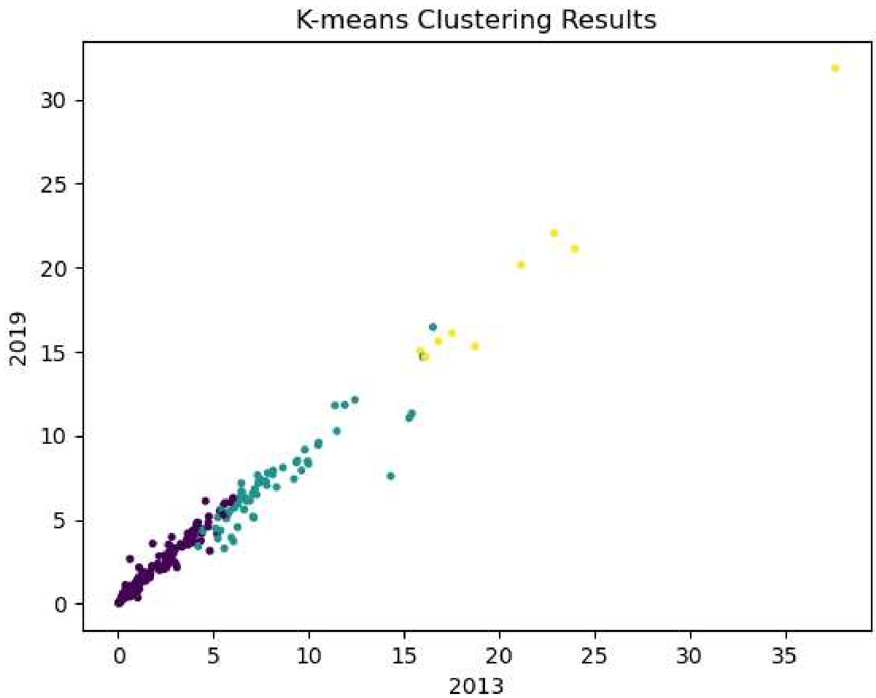


Applying K-means Clustering

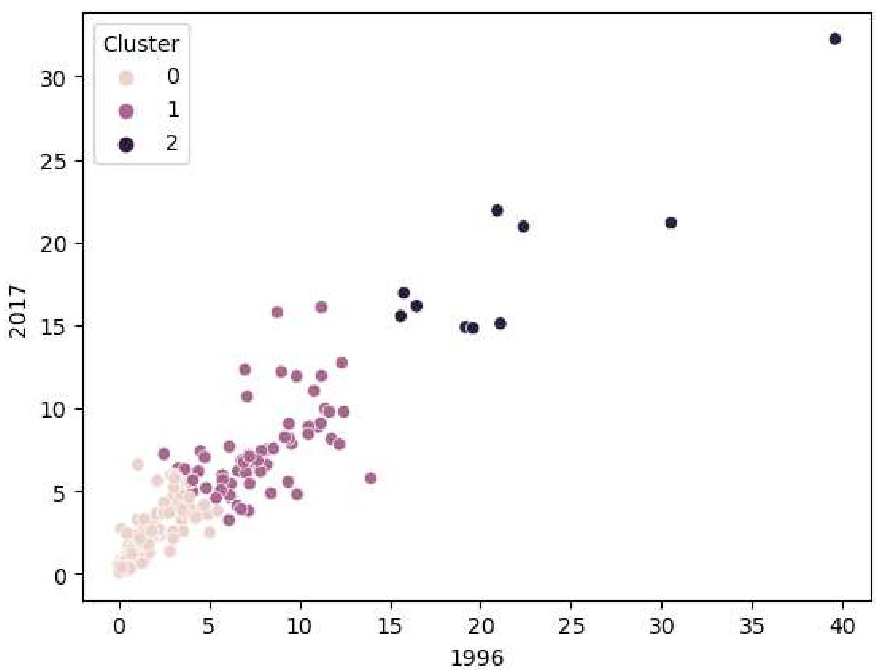
In [13]:

```
optimal_k = 3

# Apply K-means clustering

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

In [14]:
```
dot_size = 7  # Adjust this value based on your preference

# Visualize the results using a scatter plot with smaller dots
plt.scatter(df['2013'], df['2019'], c=df['Cluster'], cmap='viridis', s=dot_size)
plt.title('K-means Clustering Results')
plt.xlabel('2013')
plt.ylabel('2019')
plt.show()

```

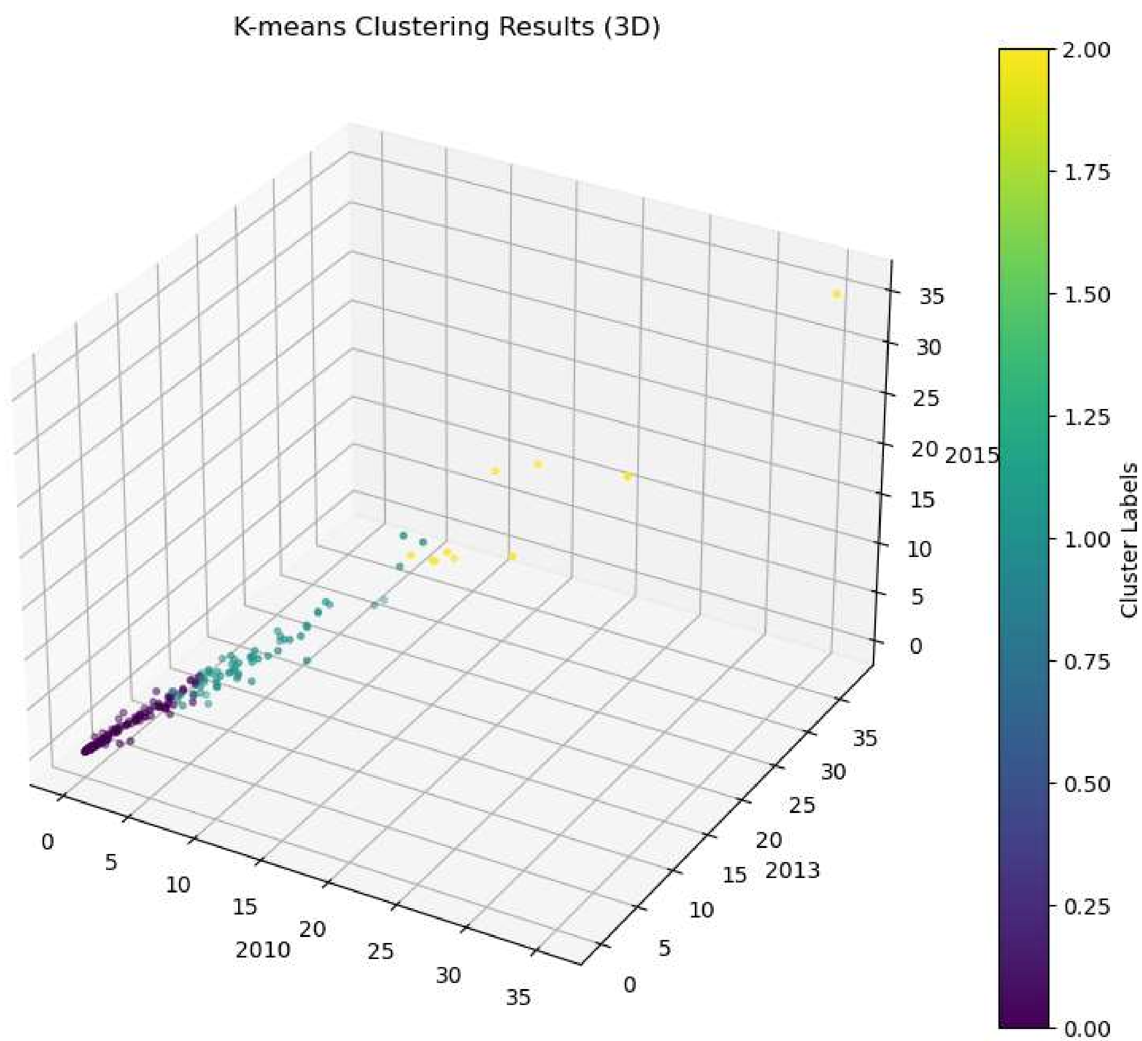


In [15]:
```
sns.scatterplot(x='1996', y='2017', hue='Cluster', data=df)
plt.show()
```



In [16]:

```
from mpl_toolkits.mplot3d import Axes3D

# Assuming 'original_dataset' contains your unscaled features and 'Cluster_Labels' column
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Set a smaller size for the dots
dot_size = 7  # Adjust this value based on your preference

scatter = ax.scatter(
    df['2010'],
    df['2013'],
    df['2015'],
    c=df['Cluster'],
    cmap='viridis',
    s=dot_size  # Set the size of the dots
)

ax.set_xlabel('2010')
ax.set_ylabel('2013')
ax.set_zlabel('2015')
ax.set_title('K-means Clustering Results (3D)')

# Add a colorbar to show cluster assignments
fig.colorbar(scatter, ax=ax, label='Cluster Labels')

plt.show()

```



In [17]:
```
cluster_summary = df.groupby('Cluster').mean()
cluster_summary
```

Out[17]:

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	20
Cluster												
0	1.427813	1.398819	1.361062	1.328024	1.353047	1.379618	1.423309	1.447565	1.439647	1.437016	...	1.7893
1	8.222173	8.080053	7.869553	7.793251	7.779457	7.931050	7.834609	7.778298	7.713726	7.765690	...	8.0638
2	19.919135	20.581825	21.543791	22.253110	21.786230	22.105216	22.824967	22.609215	22.740710	22.722836	...	21.1474

3 rows × 30 columns

In [18]:
```
from sklearn.model_selection import train_test_split

common_columns = ['1992', '1993', '1994', '1995', '1996', '1997', '1998', '1999', '2000',
                  '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009',
                  '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018',
                  '2019', '2020']

# Subset both data frames to include only common columns
df1_common = df1[common_columns]
df_common = df[common_columns + ['Country Name', 'Cluster']]  # Add other relevant columns

```

In [19]:
```
df_common
```

Out[19]:

	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	...	2013	2014	
0	0.907936	0.909550	0.913413	0.933001	0.943200	0.962203	0.963157	0.902134	0.891352	0.958883	...	1.001154	1.013758	0.
1	0.126517	0.109106	0.096638	0.088781	0.082267	0.075559	0.071270	0.058247	0.055167	0.055293	...	0.298088	0.283692	0.
2	0.558013	0.513859	0.462384	0.492656	0.554305	0.540062	0.506709	0.502905	0.521689	0.533552	...	0.481623	0.493505	0.
3	0.544413	0.710961	0.839266	0.914265	1.073630	1.086325	1.091173	1.109791	0.988416	0.941818	...	1.031044	1.091497	1.
4	0.689644	0.644008	0.649938	0.612055	0.621206	0.469831	0.576804	0.960297	1.031568	1.056868	...	1.656390	1.795712	1.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
232	0.606011	0.656505	0.597318	0.666659	0.744144	0.723075	0.809934	0.796330	0.807574	0.876408	...	0.983800	1.027474	1.
233	0.632544	0.570608	0.600495	0.654007	0.649987	0.680397	0.707366	0.776116	0.806846	0.839206	...	1.031167	0.988347	0.
234	5.717823	5.795258	5.826213	6.007616	6.136002	6.372629	6.459824	6.001786	6.076553	6.783723	...	8.116435	8.191153	7.
235	0.352722	0.304005	0.252979	0.245217	0.197921	0.250242	0.231850	0.183344	0.182709	0.180071	...	0.278215	0.297755	0.
236	1.735620	1.581818	1.469850	1.408363	1.329556	1.216829	1.218623	1.350076	1.147382	1.137220	...	0.901248	0.866838	0.

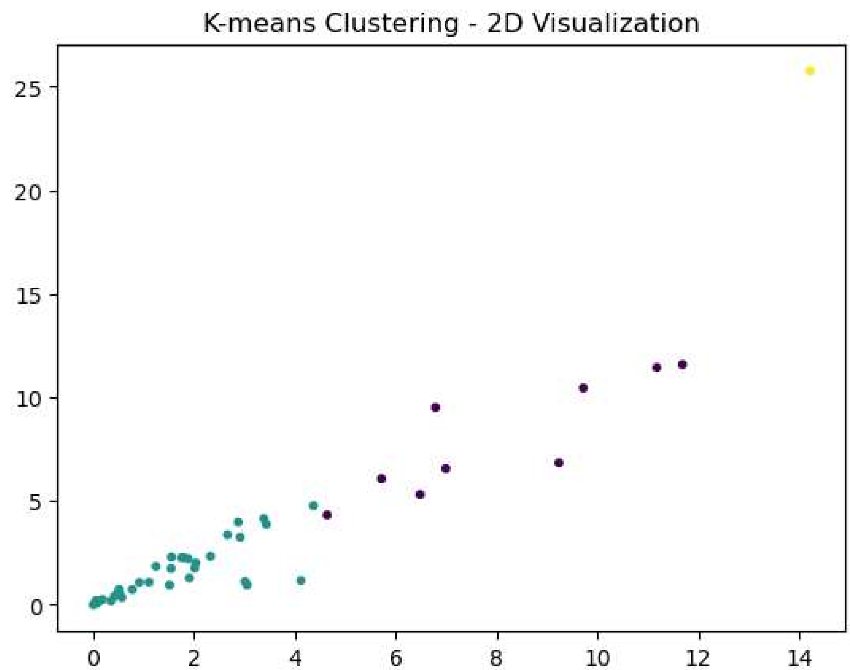
237 rows × 31 columns

In [20]:

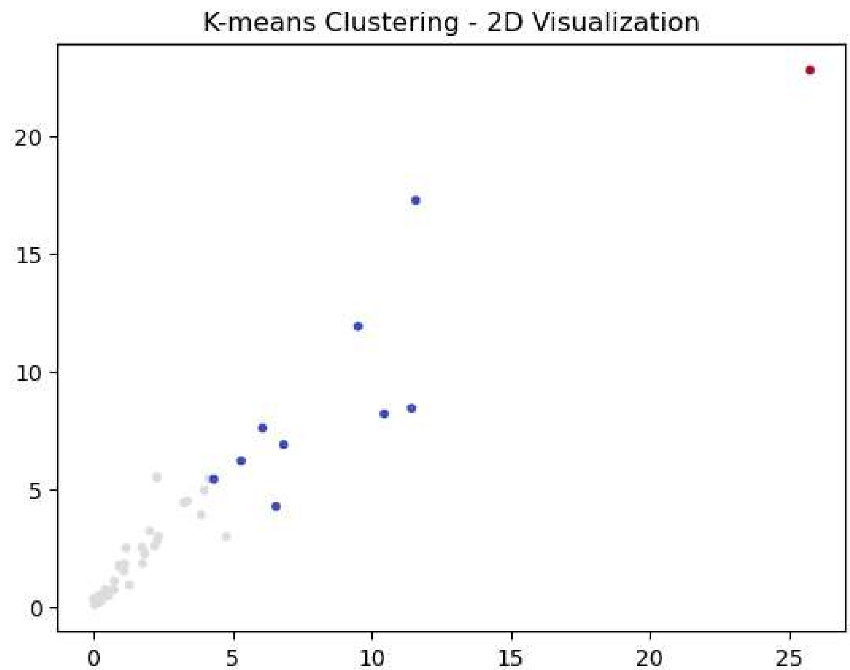
```
features = df_common.drop(['Country Name', 'Cluster'], axis=1)
clusters = df_common['Cluster']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(features, clusters, test_size=0.2, random_state=4

# Train K-means on the training data
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_train)

# Predict clusters on the test data
test_clusters = kmeans.predict(X_test)
```

In [37]:

```
# 2D Scatter Plot

plt.scatter(X_test['1992'], X_test['2000'], c=test_clusters, cmap='viridis', s=10)  # Adjust 's' valu
plt.title('K-means Clustering - 2D Visualization')
plt.show()
```

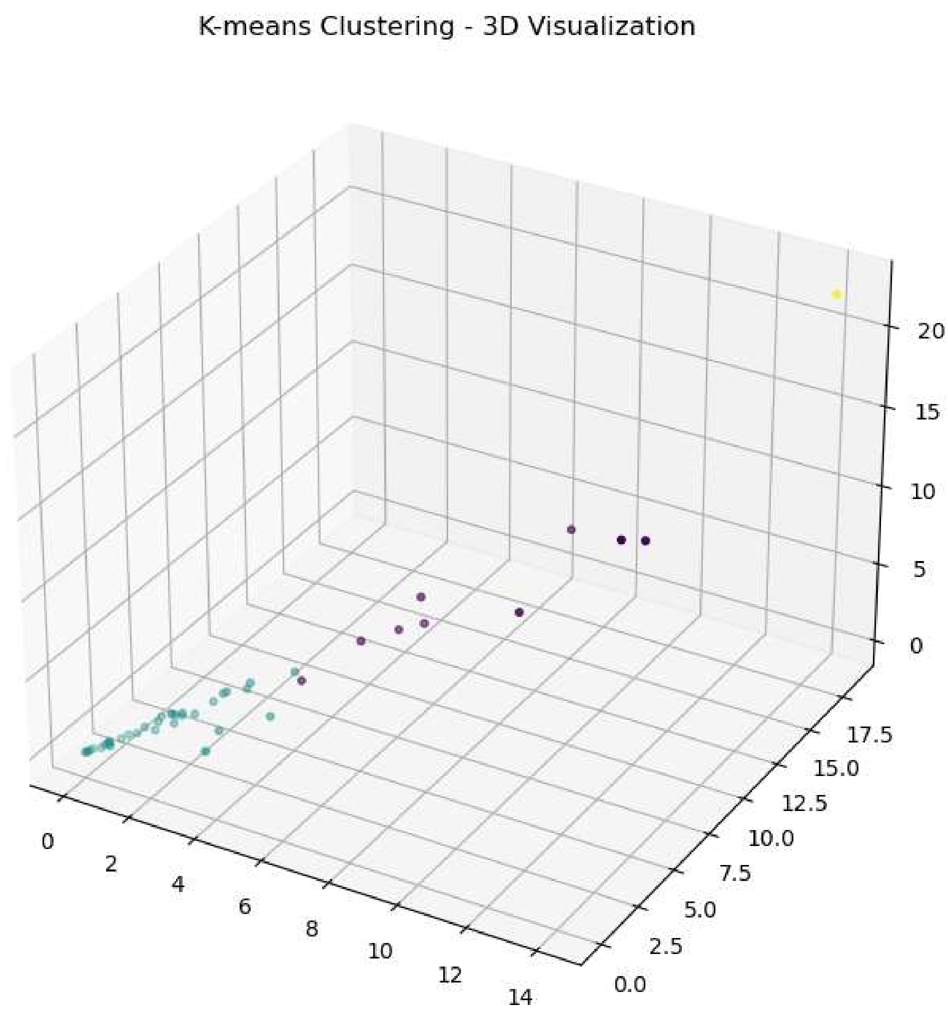


In [35]:
```
# 2D Scatter Plot -2

plt.scatter(X_test['2000'], X_test['2015'], c=test_clusters, cmap='coolwarm', s=10)  # Adjust 's' valu
plt.title('K-means Clustering - 2D Visualization')
plt.show()

```



In [22]:

```
from mpl_toolkits.mplot3d import Axes3D  # for 3D plots

# 3D Scatter Plot

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_test['1992'], X_test['1993'], X_test['1994'], c=test_clusters, cmap='viridis', s=10)  # 
ax.set_title('K-means Clustering - 3D Visualization')
plt.show()
```



Evaluate Cohesion and Separation

In [23]:

```
from sklearn.metrics import silhouette_score

# kmeans is the trained K-means model
inertia = kmeans.inertia_
silhouette = silhouette_score(X_test, test_clusters)

print(f"Inertia: {inertia}")
print(f"Silhouette Score: {silhouette}")

```

```
Inertia: 31720.482657475437
Silhouette Score: 0.638077851285394
```

Inspect Cluster Sizes

In [24]:
```
cluster_sizes = X_test.groupby(test_clusters).size()
print("Cluster Sizes:")
print(cluster_sizes)

```

```
Cluster Sizes:
0     9
1    38
2     1
dtype: int64
```

In [25]:
```
new_data = df_common
```

In [26]:
```
df1 = df1.drop(['1991'], axis=1)
```

Make Predictions on New Data

In [27]:
```
new_data_clusters = kmeans.predict(df1)
```

Analyze Cluster Centers

In [28]:
```
cluster_centers = kmeans.cluster_centers_
print("Cluster Centers:")
print(cluster_centers)
```

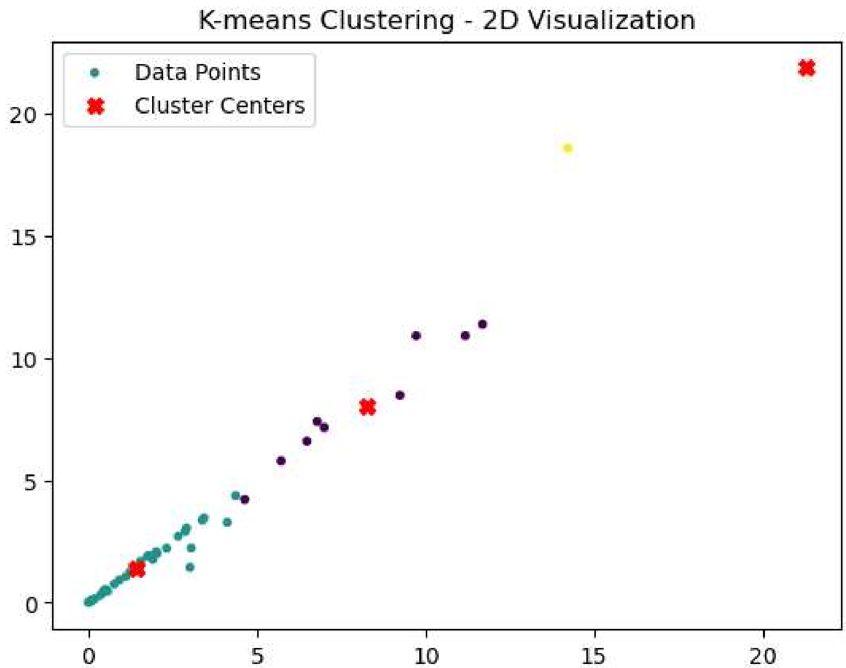
```
Cluster Centers:
[[ 8.28426524  8.01993061  7.94458669  7.92316716  8.0840896   7.97769568
   7.9330942   7.87665938  7.90147893  8.02057112  8.06742695  8.35174086
   8.35226625  8.32589279  8.46491597  8.5183357   8.37388878  7.83690559
   8.16436511  8.0285668   7.99279307  7.93501432  7.59839331  7.39217245
   7.31832485  7.34207293  7.29504353  6.99635447  6.37262892]
 [ 1.39321741  1.37022622  1.34656531  1.37564091  1.40225317  1.45007732
   1.47057223  1.47038827  1.45724774  1.50414889  1.49040285  1.52970369
   1.61564352  1.65816131  1.70811441  1.74345729  1.76607479  1.73851557
   1.79274071  1.83734154  1.86149989  1.84825408  1.85573135  1.85640443
   1.8901272   1.90979038  1.94234371  1.97078468  1.81887419]
 [21.28866745 21.87190048 22.24282654 21.77363312 22.23722059 23.00782918
  22.5484461  22.56034648 22.38565228 22.38657753 22.73653289 23.20446379
  23.37809936 23.19716465 23.19658392 22.32580692 21.75937132 20.29216765
  20.49594754 20.55339329 20.53951641 20.2962328  19.96881982 19.30893648
  18.99434427 18.64751924 18.44006864 18.39598242 18.06844054]]
```

In [29]:
```
df.columns
```

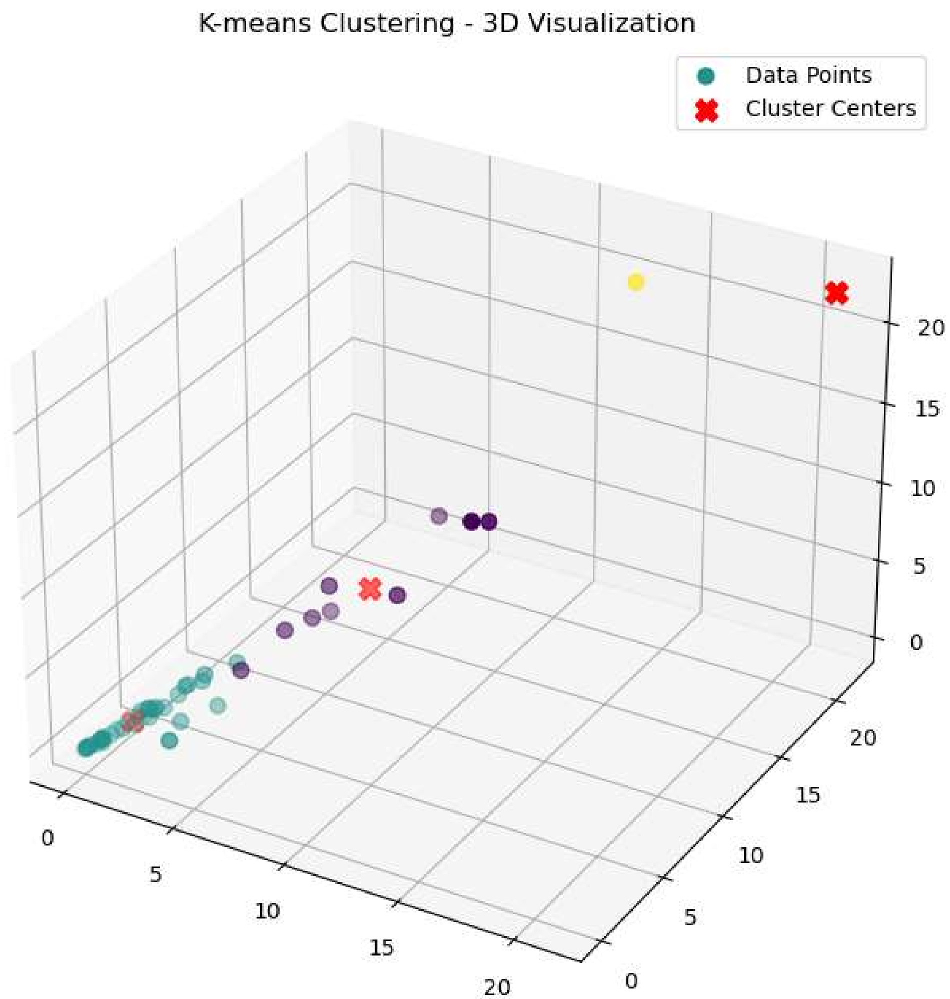
Out[29]:
```
Index(['Country Name', 'Indicator Name', '1991', '1992', '1993', '1994',
       '1995', '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003',
       '2004', '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012',
       '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020',
       'Cluster'],
      dtype='object')
```

In [30]:

```
# 2D Scatter Plot with smaller dots and cross

plt.scatter(X_test['1992'], X_test['1993'], c=test_clusters, cmap='viridis', s=10, label='Data Points
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='X', s=50, c='red', label='Cluster C
plt.title('K-means Clustering - 2D Visualization')
plt.legend()
plt.show()
```



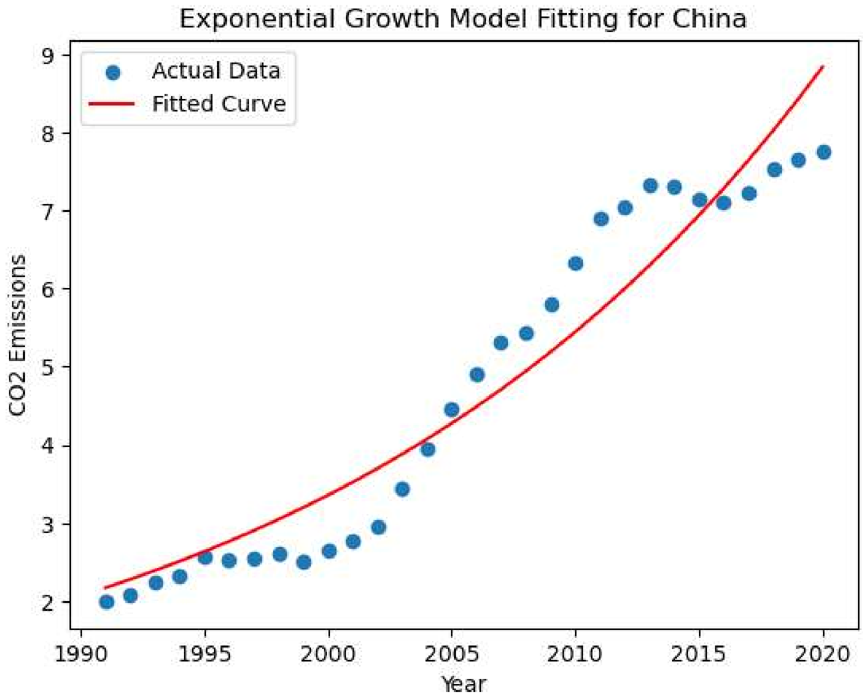
In [31]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Import this for 3D scatter plot

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_test['1992'], X_test['1993'], X_test['1994'], c=test_clusters, cmap='viridis', s=50, lab
ax.scatter(cluster_centers[:, 0], cluster_centers[:, 1], cluster_centers[:, 2], marker='X', s=100, c=
ax.set_title('K-means Clustering - 3D Visualization')
ax.legend()
plt.show()

```



Model Fitting

In [32]:
```
selected_country = 'China'
co2_emissions = df[df['Country Name'] == selected_country].loc[:, '1991':'2000'].values.flatten()
```

In [33]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Example: Replace 'United States' with the country for which you want to fit the model
selected_country = 'China'

# Extract relevant rows for the selected country
country_data = df[df['Country Name'] == selected_country]

# Extract the years and CO2 emissions values
years = country_data.columns[2:-1].astype(int)  # assuming '1991' to '2020' columns
co2_emissions = country_data.iloc[:, 2:-1].values.flatten()

# Define the exponential growth model
def exponential_growth(t, a, b):
    return a * np.exp(b * (t - 1991))

# 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, co2_emissions)

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

# Plot the original data and the fitted curve
plt.scatter(years, co2_emissions, label='Actual Data')
plt.plot(years, predicted_values, color='red', label='Fitted Curve')
plt.xlabel('Year')
plt.ylabel('CO2 Emissions')
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): [2.17151711 0.04839454]
Confidence Intervals (95%):
Lower Bound: [1.88036033 0.04232672]
Upper Bound: [2.46267388 0.05446237]
```

In [ ]:
```
1
```