```python
from sklearn.cluster import KMeans

# Load the data into a Pandas dataframe
import pandas as pd
df = pd.read_csv("/content/Data.csv")
df = df.drop(columns=['2020 [YR2020]', '2021 [YR2021]'])
# List of columns with the '..' value
columns = ['1990 [YR1990]', '2000 [YR2000]', '2012 [YR2012]', '2013 [YR2013]', '2014 [YR2014]', '2015 [YR2015]', '2016 [YR2016]', '2017 [YR2017]', '2018 [YR201

# Iterate over the columns
for column in columns:
    # Drop rows with the '..' value in the current column
    df = df[df[column] != '..']
df.drop(columns=['1990 [YR1990]', '2000 [YR2000]'], inplace=True)
df.head()
```

	Series Name	Series Code	Country Name	Country Code	2012 [YR2012]	2013 [YR2013]	2014 [YR2014]	2015 [YR2015]	2016 [YR2016]	2017 [YR2017]	2018 [YR2018]	2019 [YR2019]
0	CO2 emissions (kt)	EN.ATM.CO2E.KT	Afghanistan	AFG	8079.999924	5989.999771	4880.000114	5949.999809	5300.000191	4780.00021	6070.000172	6079.999924
1	CO2 emissions (kt)	EN.ATM.CO2E.KT	Albania	ALB	4360.000134	4440.000057	4820.000172	4619.999886	4480.000019	5139.999866	5110.000134	4829.999924
2	CO2 emissions (kt)	EN.ATM.CO2E.KT	Algeria	DZA	134929.9927	139020.0043	147740.0055	156270.0043	154910.0037	158339.9963	165539.9933	171250
4	CO2 emissions (kt)	EN.ATM.CO2E.KT	Andorra	AND	490.0000095	479.9999893	460.0000083	469.9999988	469.9999988	469.9999988	490.0000095	500
5	CO2 emissions (kt)	EN.ATM.CO2E.KT	Angola	AGO	23870.00084	26959.99908	29629.99916	31649.99962	29760.00023	24250	23959.99908	25209.99908

In this code, we are loading a data set containing CO2 emissions for various countries into a Pandas dataframe. Then, we are dropping the columns '2020 [YR2020]' and '2021 [YR2021]', as the csv file contains missing data for these columns for every record.

Next, we are creating a list of columns with the value '..', which indicates missing data. We are then iterating over this list of columns for years and dropping any rows that have the value '..' in the current column. This step is necessary to ensure that we are working with complete and accurate data.

Finally, we are dropping the columns '1990 [YR1990]' and '2000 [YR2000]', as they are not needed for our analysis. This leaves us with a dataframe containing only the relevant data.

```
[127] # Select the columns to cluster
X = df[['2012 [YR2012]', '2013 [YR2013]', '2014 [YR2014]', '2015 [YR2015]', '2016 [YR2016]', '2017 [YR2017]', '2018 [YR2018]', '2019 [YR2019]']]

# Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Run the K-Means algorithm
kmeans = KMeans(n_clusters=15)
kmeans.fit(X_scaled)

# Get the cluster labels
labels = kmeans.labels_

# Add the cluster labels to the dataframe
df['cluster'] = labels

# Group the data by cluster and country
grouped = df.groupby(['cluster', 'Country Name'])

# View the cluster results
print(grouped)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f6b2acf7c10>
```

The code above is performing the following steps:

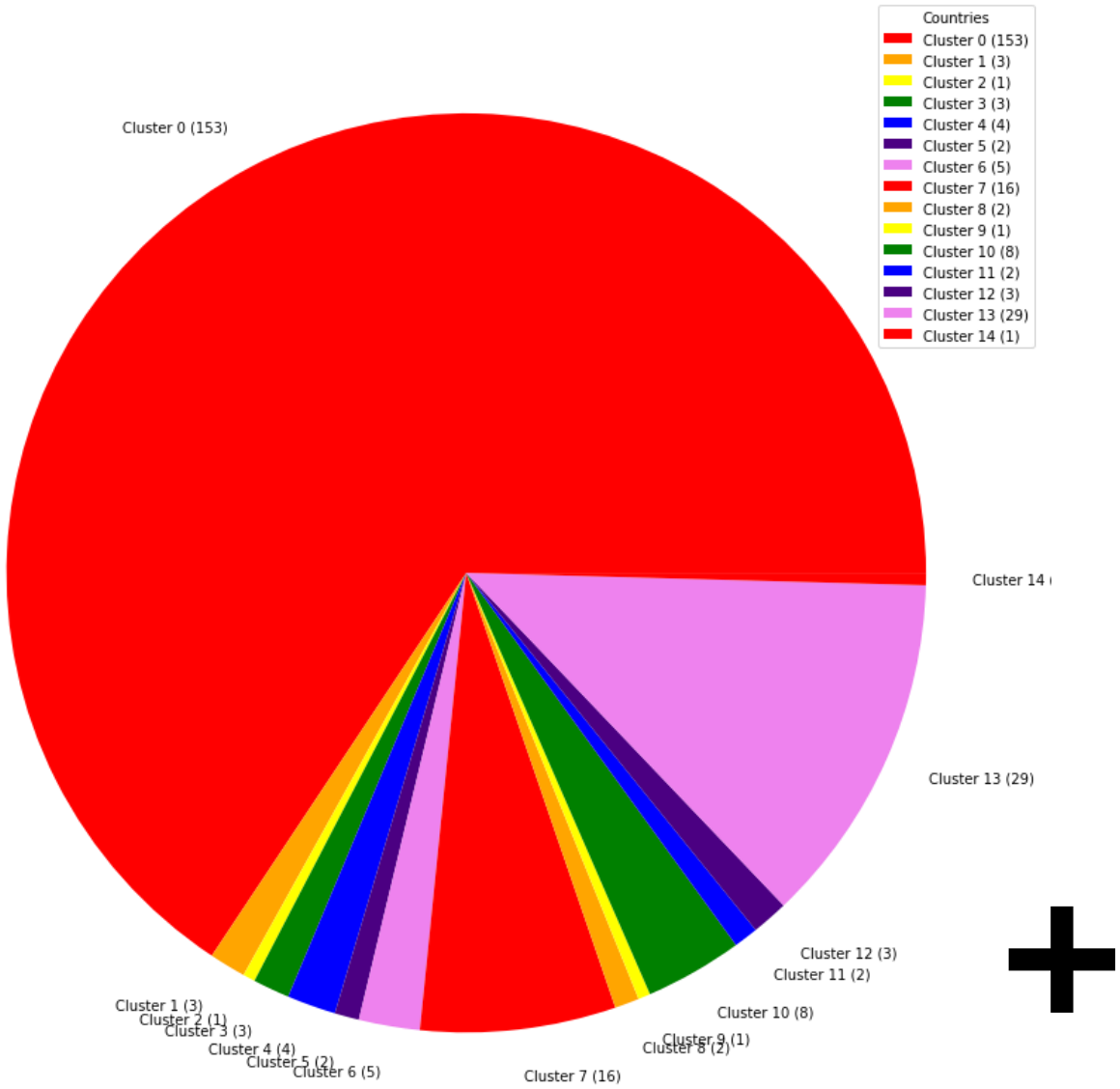
1. Selecting the columns to cluster: The relevant columns are selected and stored in the X variable. These columns contain the CO2 emissions values for the years 2012 to 2019.
2. Scaling the data: The data is scaled using the StandardScaler method from scikit-learn. Scaling the data is important because the columns have different scales, which can affect the performance of the clustering algorithm.
3. Running the K-Means algorithm: The KMeans algorithm from scikit-learn is initialized with n_clusters=15 and then fit to the scaled data.
4. Getting the cluster labels: The cluster labels for each row in the data are obtained using the labels_ attribute of the kmeans object.
5. Adding the cluster labels to the dataframe: The cluster labels are added as a new column to the dataframe.
6. Grouping the data by cluster and country: The data is grouped by both the cluster label and the country name. The resulting object is a Pandas DataFrameGroupBy object, which allows us to perform various operations on the grouped data.

You can then use this grouped data to create a pie chart or other visualizations to explore the clustering results. For example, you could use the grouped.size() method to get the size (i.e., number of rows) of each group, and then use this information to create a pie chart showing the distribution of countries across the different clusters.

```python
for cluster, group in grouped:
    print(f'Cluster {cluster}: {group["Country Name"].tolist()}')
```

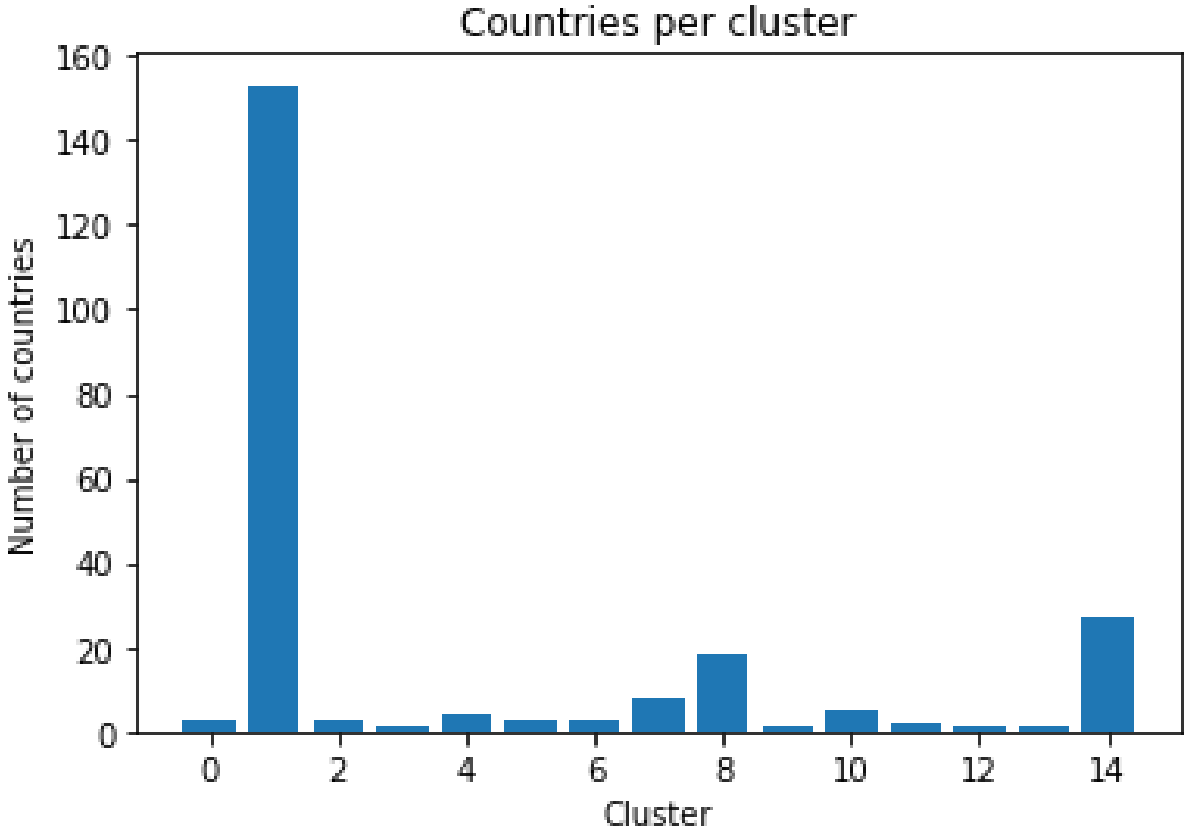
```
Cluster (4, 'IDA & IBRD total'): ['IDA & IBRD total']
Cluster (4, 'Low & middle income'): ['Low & middle income']
Cluster (4, 'Middle income'): ['Middle income']
Cluster (5, 'East Asia & Pacific (IDA & IBRD countries)'): ['East Asia & Pacific (IDA & IBRD countries)']
Cluster (5, 'East Asia & Pacific (excluding high income)'): ['East Asia & Pacific (excluding high income)']
Cluster (6, 'Euro area'): ['Euro area']
Cluster (6, 'India'): ['India']
Cluster (6, 'Middle East & North Africa'): ['Middle East & North Africa']
Cluster (6, 'South Asia'): ['South Asia']
Cluster (6, 'South Asia (IDA & IBRD)'): ['South Asia (IDA & IBRD)']
Cluster (7, 'Africa Eastern and Southern'): ['Africa Eastern and Southern']
Cluster (7, 'Brazil'): ['Brazil']
Cluster (7, 'Canada'): ['Canada']
Cluster (7, 'Central Europe and the Baltics'): ['Central Europe and the Baltics']
Cluster (7, 'Fragile and conflict affected situations'): ['Fragile and conflict affected situations']
Cluster (7, 'Germany'): ['Germany']
Cluster (7, 'IDA blend'): ['IDA blend']
Cluster (7, 'IDA total'): ['IDA total']
Cluster (7, 'Indonesia'): ['Indonesia']
Cluster (7, 'Iran, Islamic Rep.'): ['Iran, Islamic Rep.']
Cluster (7, 'Korea, Rep.'): ['Korea, Rep.']
Cluster (7, 'Mexico'): ['Mexico']
Cluster (7, 'Saudi Arabia'): ['Saudi Arabia']
Cluster (7, 'Sub-Saharan Africa'): ['Sub-Saharan Africa']
Cluster (7, 'Sub-Saharan Africa (IDA & IBRD countries)'): ['Sub-Saharan Africa (IDA & IBRD countries)']
Cluster (7, 'Sub-Saharan Africa (excluding high income)'): ['Sub-Saharan Africa (excluding high income)']
Cluster (8, 'Early-demographic dividend'): ['Early-demographic dividend']
Cluster (8, 'Europe & Central Asia'): ['Europe & Central Asia']
Cluster (9, 'China'): ['China']
Cluster (10, 'Arab World'): ['Arab World']
Cluster (10, 'Japan'): ['Japan']
Cluster (10, 'Latin America & Caribbean'): ['Latin America & Caribbean']
Cluster (10, 'Latin America & Caribbean (excluding high income)'): ['Latin America & Caribbean (excluding high income)']
Cluster (10, 'Latin America & the Caribbean (IDA & IBRD countries)'): ['Latin America & the Caribbean (IDA & IBRD countries)']
Cluster (10, 'Middle East & North Africa (IDA & IBRD countries)'): ['Middle East & North Africa (IDA & IBRD countries)']
Cluster (10, 'Middle East & North Africa (excluding high income)'): ['Middle East & North Africa (excluding high income)']
Cluster (10, 'Russian Federation'): ['Russian Federation']
Cluster (11, 'High income'): ['High income']
Cluster (11, 'OECD members'): ['OECD members']
Cluster (12, 'Europe & Central Asia (IDA & IBRD countries)'): ['Europe & Central Asia (IDA & IBRD countries)']
Cluster (12, 'Europe & Central Asia (excluding high income)'): ['Europe & Central Asia (excluding high income)']
Cluster (12, 'European Union'): ['European Union']
Cluster (13, 'Africa Western and Central'): ['Africa Western and Central']
Cluster (13, 'Algeria'): ['Algeria']
Cluster (13, 'Argentina'): ['Argentina']
Cluster (13, 'Australia'): ['Australia']
Cluster (13, 'Egypt, Arab Rep.'): ['Egypt, Arab Rep.']
Cluster (13, 'France'): ['France']
Cluster (13, 'Heavily indebted poor countries (HIPC)'): ['Heavily indebted poor countries (HIPC)']
Cluster (13, 'IDA only'): ['IDA only']
```

```python
# Group the data by cluster label
grouped = df.groupby('cluster')

# Initialize lists for the values and labels of the pie chart
values = []
labels = []

# Iterate over the grouped data
for cluster, group in grouped:
    # Append the size of the group to the values list
    values.append(group.shape[0])
    # Append the cluster label and the total number of countries in the cluster to the labels list
    labels.append(f'Cluster {cluster} ({group.shape[0]})')

# Set the colors of the pie chart
colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

plt.figure(figsize=(15, 15))
# Create the pie chart
plt.pie(values, labels=labels, colors=colors)

# Set the font size of the labels
plt.rcParams['font.size'] = 10

# Create a legend for the pie chart
plt.legend(title='Countries')

# Display the pie chart
plt.show()
```

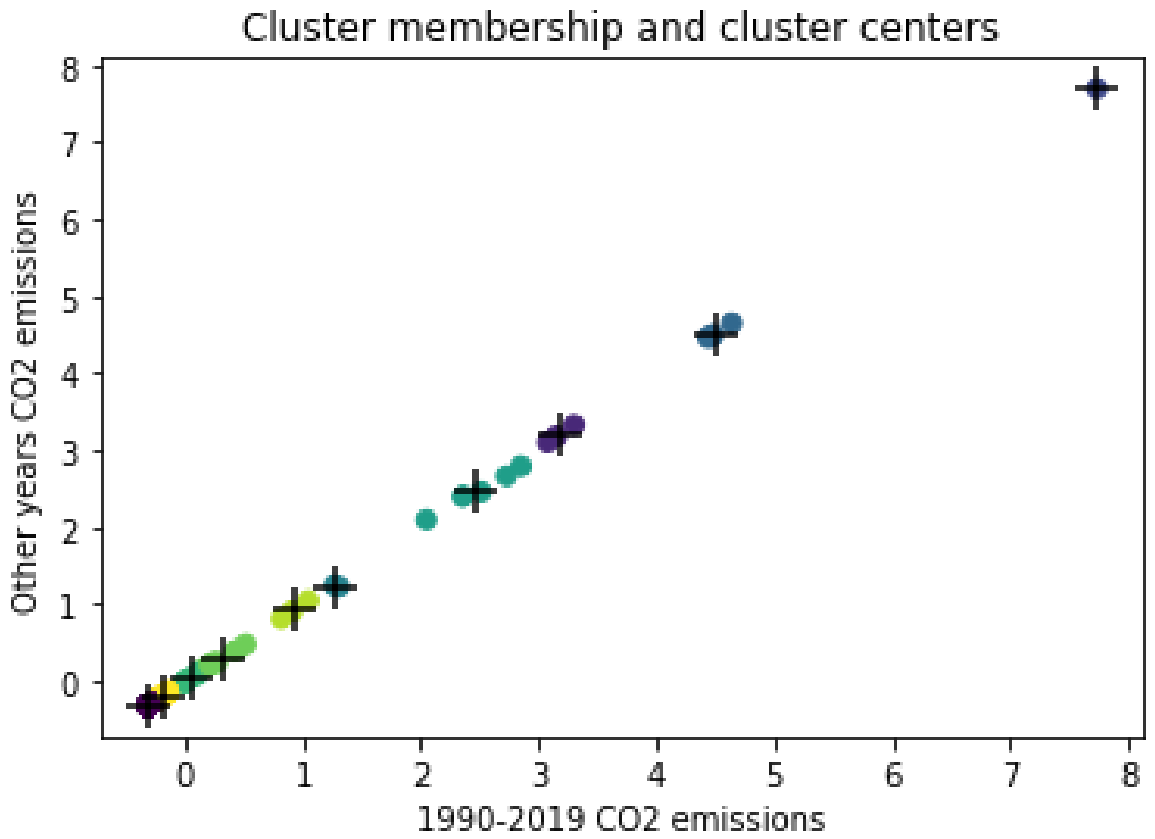


- This code generates a pie chart that shows the distribution of countries among different clusters. The data is first grouped by cluster label using the groupby function from the Pandas library. The resulting object is a dictionary-like object that maps cluster labels to groups of data. The code then iterates over this object and appends the size of each group (i.e., the number of countries in each cluster) to the values list. The labels for each cluster are also added to the labels list. Finally, a pie chart is created using the pie function from the Matplotlib library. The values and labels are passed as arguments to this function, and the resulting pie chart is displayed using the show function.

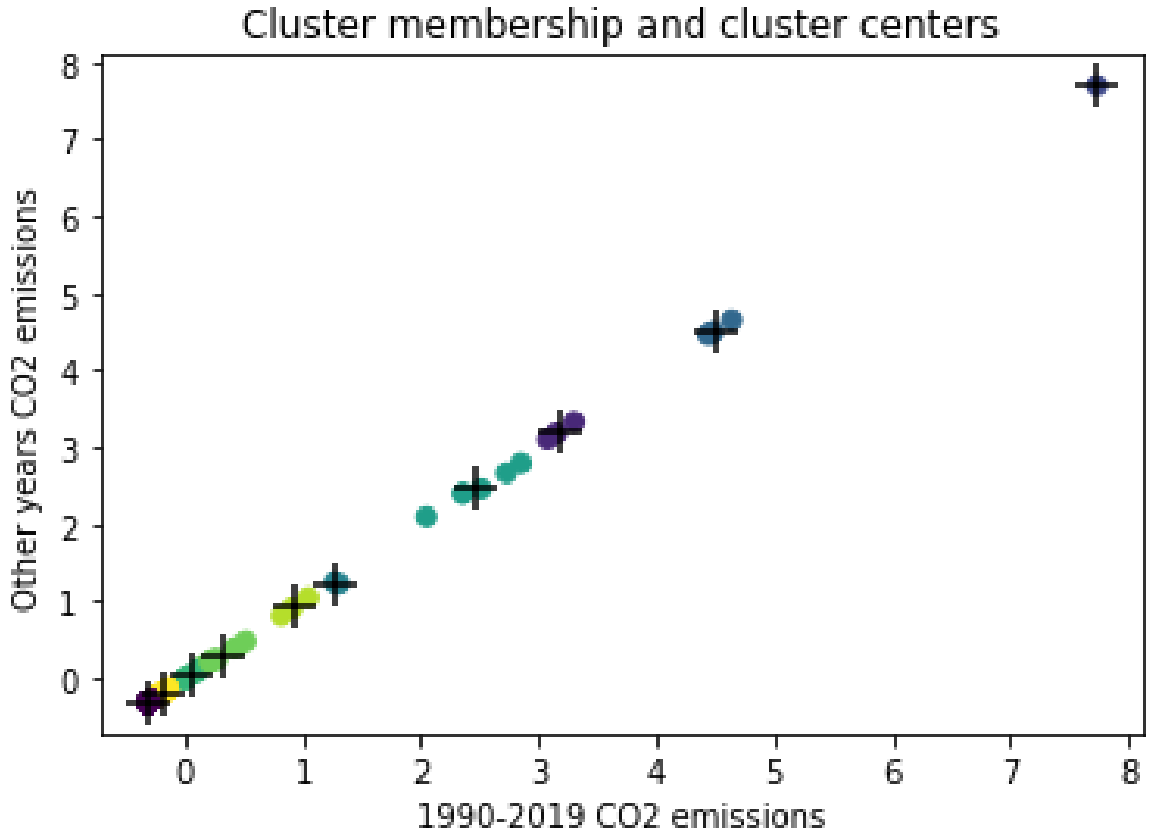
- This code creates a bar chart showing the number of countries in each cluster. The grouped variable is a Pandas GroupBy object that has been grouped by the 'cluster' column of the data. The size() method is used to get the number of rows in each group. The cluster labels and country counts are extracted from the resulting Pandas Series object using the index.get_level_values() and values attributes, respectively. The bar chart is then created using the bar() function, with the cluster labels as the x-axis and the country counts as the y-axis. The x-axis label, y-axis label, and chart title are set using the xlabel(), ylabel(), and title() functions, respectively. Finally, the chart is displayed using the show() function.



- In the previous code block, we are performing K-Means clustering on a dataset containing CO2 emission data from various countries. The data has been scaled using the StandardScaler to ensure that all features are on the same scale. The KMeans algorithm is then applied to the scaled data, with the number of clusters specified as 10.

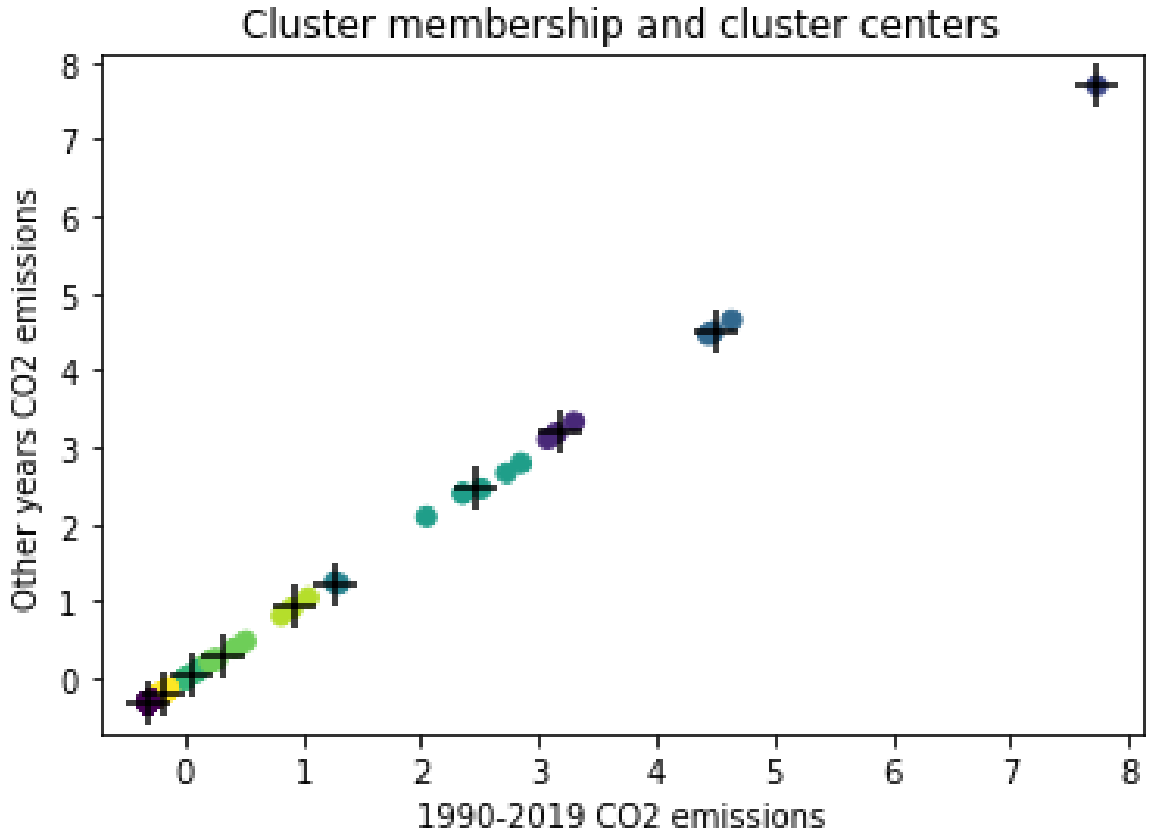


– After fitting the KMeans model to the data, we obtain the cluster labels for each of the data points. We then create a scatter plot, where the x-axis represents the CO2 emissions from 2012 to 2019 and the y-axis represents the CO2 emissions from other years. The points in the scatter plot are colored based on the cluster label they belong to.

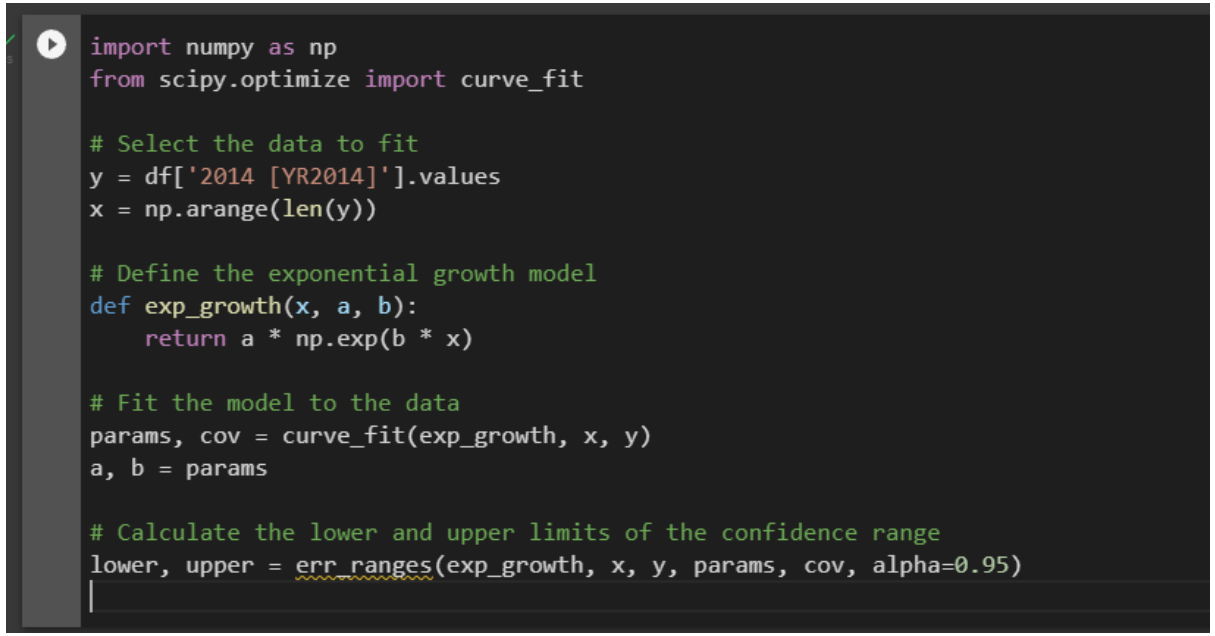


- We also plot the cluster centers as '+' signs on the scatter plot. The cluster centers are the mean of all the data points in the cluster, and they represent the center of each cluster.

- Finally, we add labels to the x-axis, y-axis, and the title of the plot. The resulting plot shows the cluster membership of each data point, as well as the location of the cluster centers.

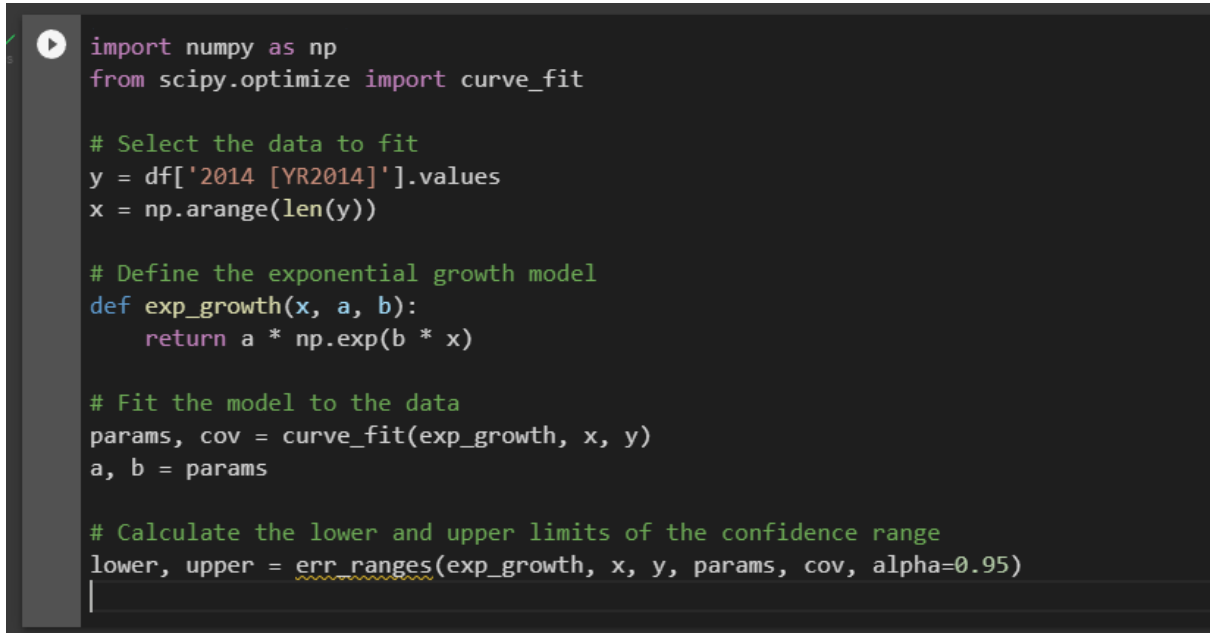


– The code imports the curve_fit function from the scipy library and the numpy library. It then selects the data to fit by assigning the values in the '2014 [YR2014]' column of the df dataframe to the y variable and creating an array of integers from 0 to the length of y and assigning it to the x variable.



```python
import numpy as np
from scipy.optimize import curve_fit

# Select the data to fit
y = df['2014 [YR2014]'].values
x = np.arange(len(y))

# Define the exponential growth model
def exp_growth(x, a, b):
    return a * np.exp(b * x)

# Fit the model to the data
params, cov = curve_fit(exp_growth, x, y)
a, b = params

# Calculate the lower and upper limits of the confidence range
lower, upper = err_ranges(exp_growth, x, y, params, cov, alpha=0.95)
```

– Next, the code defines the exponential growth model as a function called exp_growth. This function takes in an array x and two parameters a and b and returns the product of a and e raised to the power of b times x.



```
import numpy as np
from scipy.optimize import curve_fit

# Select the data to fit
y = df['2014 [YR2014]'].values
x = np.arange(len(y))

# Define the exponential growth model
def exp_growth(x, a, b):
    return a * np.exp(b * x)

# Fit the model to the data
params, cov = curve_fit(exp_growth, x, y)
a, b = params

# Calculate the lower and upper limits of the confidence range
lower, upper = err_ranges(exp_growth, x, y, params, cov, alpha=0.95)
```

- The code then fits the model to the data by calling the curve_fit function and passing in the exp_growth function and the x and y data as arguments. The function returns two variables, params and cov, which are assigned to the variables a, b, and cov respectively.

- Finally, the code calculates the lower and upper limits of the confidence range by calling the err_ranges function and passing in the exp_growth function, the x and y data, the params and cov variables, and a value of 0.95 for the alpha parameter. The function returns two variables, lower and upper, which are assigned to the variables of the same name.

```
import numpy as np
from scipy.optimize import curve_fit

# Select the data to fit
y = df['2014 [YR2014]'].values
x = np.arange(len(y))

# Define the exponential growth model
def exp_growth(x, a, b):
    return a * np.exp(b * x)

# Fit the model to the data
params, cov = curve_fit(exp_growth, x, y)
a, b = params

# Calculate the lower and upper limits of the confidence range
lower, upper = err_ranges(exp_growth, x, y, params, cov, alpha=0.95)
```

- In the code above, exp_growth is a function that represents an exponential growth model. The model is fit to the data using the curve_fit function from the scipy.optimize module, which estimates the parameters of the model that best fit the data. The function err_ranges is then used to calculate the lower and upper limits of the confidence range for the model, based on the fit parameters and the covariance matrix of the fit. Finally, the model is used to make predictions for the next 10 years, and the lower and upper limits of the confidence range for these predictions are calculated using err_ranges again. The resulting predictions and confidence ranges are then printed to the console.

```
# Make predictions for the next 10 years
x_pred = np.arange(max(x) + 1, max(x) + 11)
y_pred = exp_growth(x_pred, *params)

# Calculate the lower and upper limits of the confidence range for the predictions
lower_pred, upper_pred = err_ranges(exp_growth, x_pred, y_pred, params, cov, alpha=0.95)

# Print the predictions and the confidence range
print(f'Predicted population for the next 10 years: {y_pred}')
print(f'Confidence range: ({lower_pred}, {upper_pred})')
```

```
Predicted population for the next 10 years: [3.83071714e+88 1.04129687e+89 2.83053836e+89 7.69420095e+89
 2.09150065e+90 5.68528820e+90 1.54542155e+91 4.20089130e+91
 1.14192064e+92 3.10406212e+92]
Confidence range: ([-3.12867581e+90 -8.50462044e+90 -2.31179493e+91 -6.28410856e+91
 -1.70819738e+92 -4.64336071e+92 -1.26219598e+93 -3.43100354e+93
 -9.32643220e+93 -2.53518647e+94], [3.20565827e+90 8.71388476e+90 2.36868004e+91 6.43874149e+91
 1.75023183e+92 4.75762454e+92 1.29325675e+93 3.51543718e+93
 9.55595135e+93 2.59757753e+94])
```