7PAM2000

Applied Data Science 1

Assignment 3:

Clustering and fitting

Github link: https://github.com/shazaib001/assignment-3-clustering-and-fitting

Table of Content

Introduction

Intro:Dataset Information

- The dataset is from the World Bank and contains information about Gross Domestic Product (GDP) per capita, which is a measure of the average economic output per person in a given area.
- The indicator code "NY.GDP.PCAP.PP.CD" stands for "GDP per capita, PPP (current international $)". PPP stands for "purchasing power parity", which adjusts for differences in cost of living across countries.
- The data can be used to study the economic performance of the countries.
- We use the data to compare the GDP per capita among the countries, and also over time.

Intro:Objective of the analysis

- To study the GDP per capita of different countries
- To find clusters of countries that have similar GDP per capita
- To fit a model to the GDP per capita data to make predictions
- To estimate the error ranges of the predictions

Intro:Objective of the analysis

- To group countries by their GDP per capita data using the K-Means clustering algorithm
- To fit the data with exponential growth model
- To explore the trends in GDP per capita among the countries
- To find patterns and insights in the data.

Clustering Results

Clustering Results

- For clustering we need normalized data of GDP per capita. So,
- Normalizes the GDP per capita data using MinMaxScaler from sklearn.
- It uses the fit_transform() method to fit the scaler on the data and transform it
- it uses '.loc[:,'2010':'2021']' to select the column from 2010 to 2021
- The normalized data is stored in a variable 'data_scaled' and will be used for clustering analysis.

```
# Normalize the data
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data.loc[:,'2010':'2021'])
```

Clustering Results

- For clustering we need normalized data of GDP per capita. So,
- Normalizes the GDP per capita data using MinMaxScaler from sklearn.
- It uses the fit_transform() method to fit the scaler on the data and transform it
- it uses '.loc[:,'2010':'2021']' to select the column from 2010 to 2021
- The normalized data is stored in a variable 'data_scaled' and will be used for clustering analysis.

```
# Normalize the data
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data.loc[:,'2010':'2021'])
```

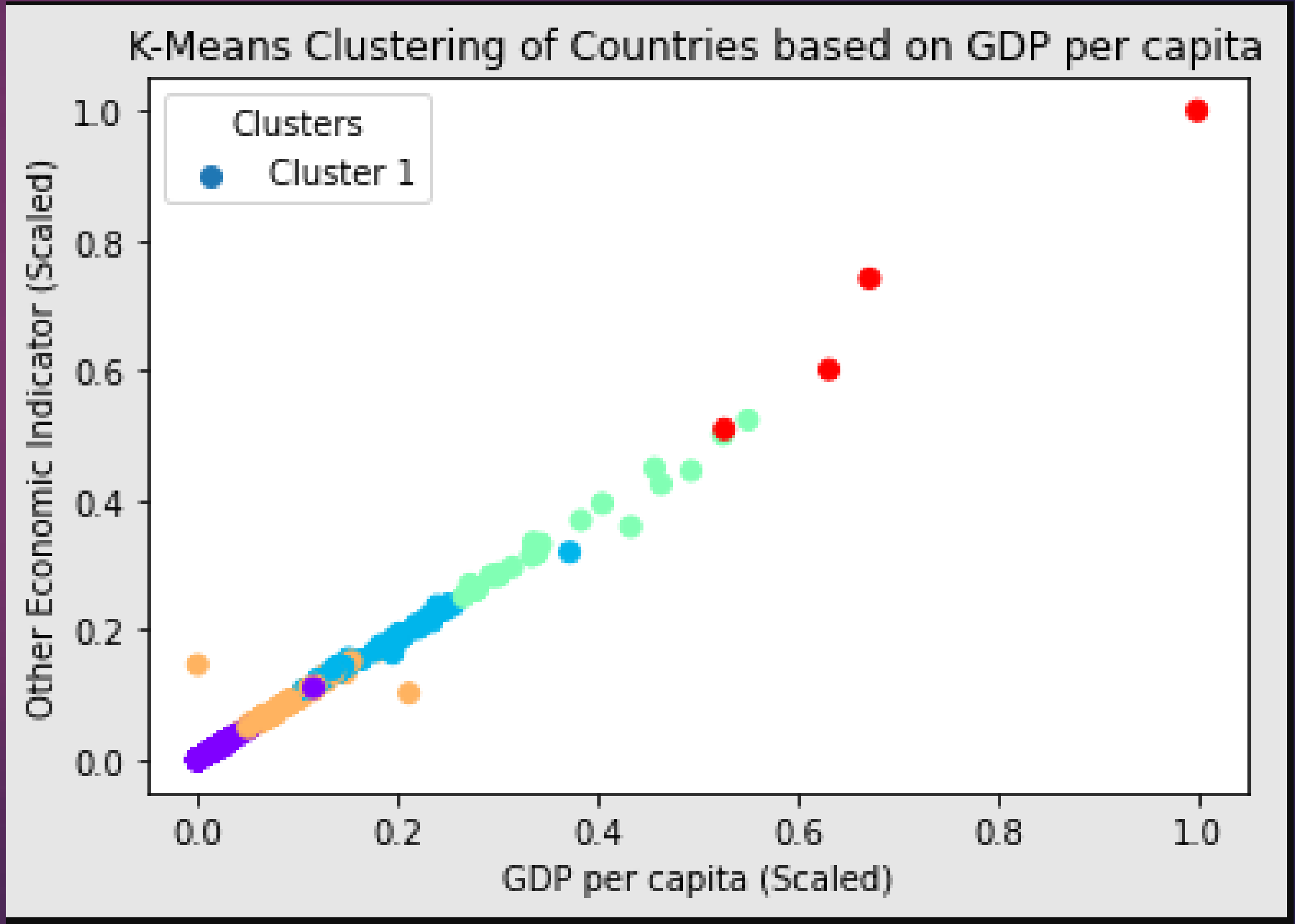
Clustering Results

- Uses the KMeans class from sklearn library to perform the clustering
- uses n_clusters=5, which means that we will create 5 clusters of countries
- then uses the fit_predict() method to fit the data and predict the cluster labels

```
# Perform K-Means clustering
kmeans = KMeans(n_clusters=5)
clusters = kmeans.fit_predict(data_scaled)
clusters = kmeans.labels_
```

Clustering Results

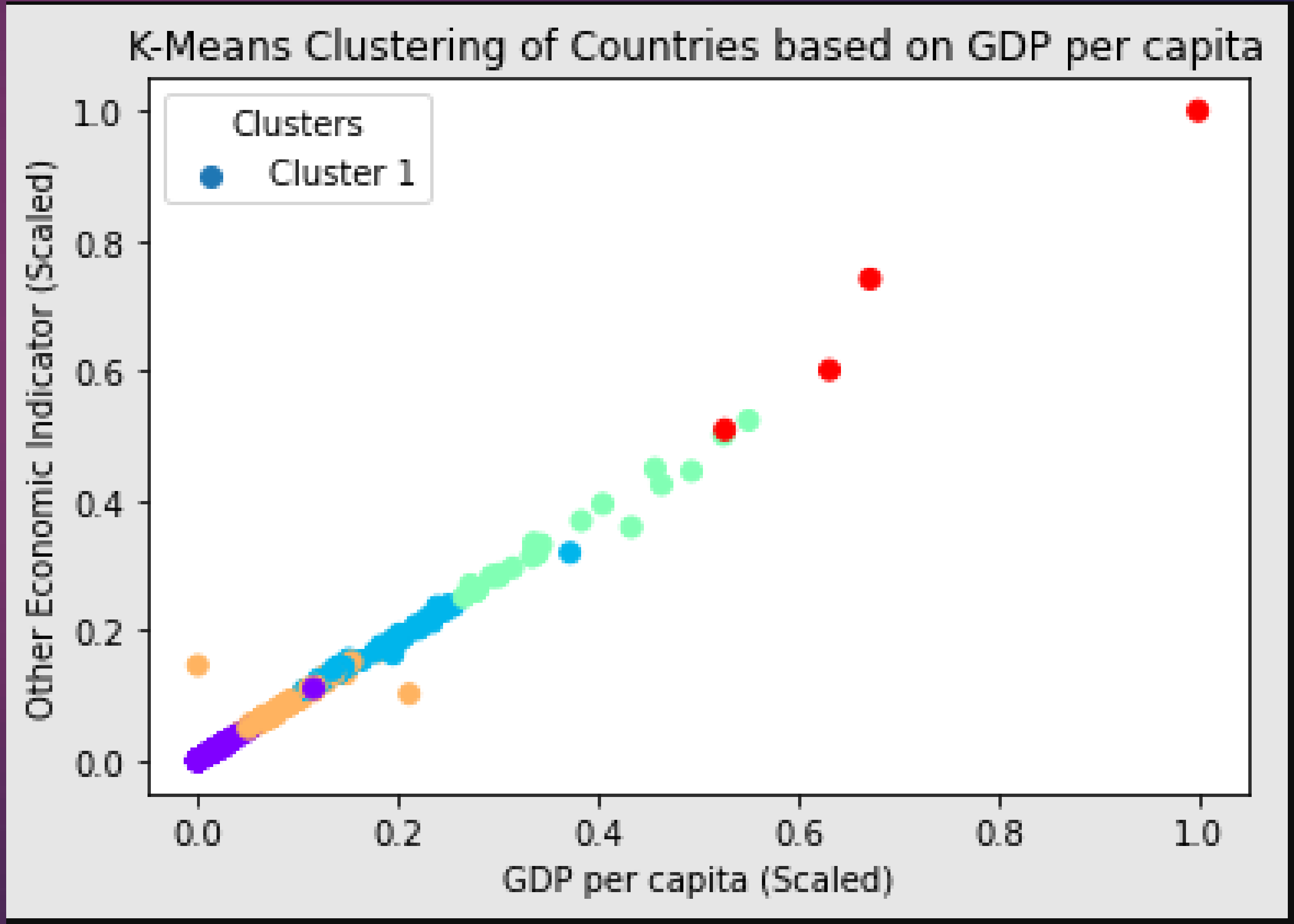
- then creates a scatter plot to visualize the clusters, where the x-axis represents GDP per capita (scaled), the y-axis represents other economic indicator (scaled), and the color of the points represents the cluster label
- The plot is labeled with a title, x-axis label, y-axis label, legend, and captions
- Plot is shown using plt.show() from pyplot.



Clustering Results

Plotting Code for Visualization:

```
plt.scatter(data_scaled[:,0], data_scaled[:,1], c=clusters, cmap='rainbow')
plt.title("K-Means Clustering of Countries based on GDP per capita")
plt.xlabel("GDP per capita (Scaled)")
plt.ylabel("Other Economic Indicator (Scaled)")
plt.legend(title='Clusters', labels=['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5'])
plt.show()
```



Model Fitting Results

Model Fitting Results

- For Model Fitting function **'exponential_growth'** is used that takes in two parameters (x, a, b) and returns the exponential growth of x with parameters a and b.
- First reshapes the x variable to ensure that it is in the correct format
- then defines x_data as the index of dataframe and y_data as the 2020 column

```
# Perform K-Means clustering
kmeans = KMeans(n_clusters=5)
clusters = kmeans.fit_predict(data_scaled)
clusters = kmeans.labels_
```

Model Fitting Results

- For Model Fitting function **'exponential_growth'** is used that takes in two parameters (x, a, b) and returns the exponential growth of x with parameters a and b.
- First reshapes the x variable to ensure that it is in the correct format
- then defines x_data as the index of dataframe and y_data as the 2020 column

```
# Fit a simple model to the data
def exponential_growth(x, a, b):

    x = x.reshape(-1)

    return a * np.exp(b * x)


x_data = np.array(data.index).reshape(-1,1)
y_data = data['2020'].values
np.isnan(x_data).any()
np.isinf(x_data).any()

params, cov = curve_fit(exponential_growth, x_data, y_data)
```

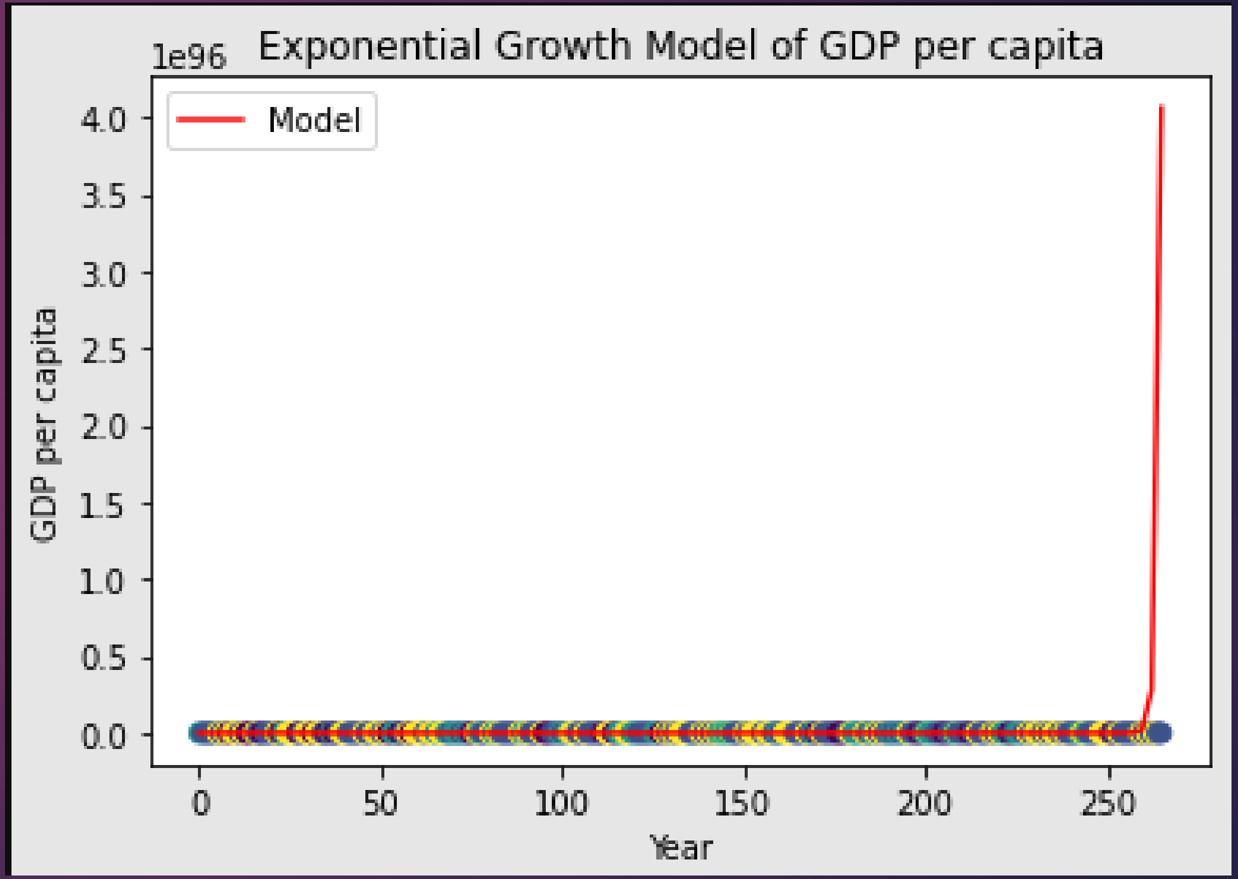
Model Fitting Results

- checks if x_data contains any NaN or Inf values and eliminates them
- then uses curve_fit function from scipy to find the best parameters for the function
- After that uses the function to predict the values and store them in y_pred variable

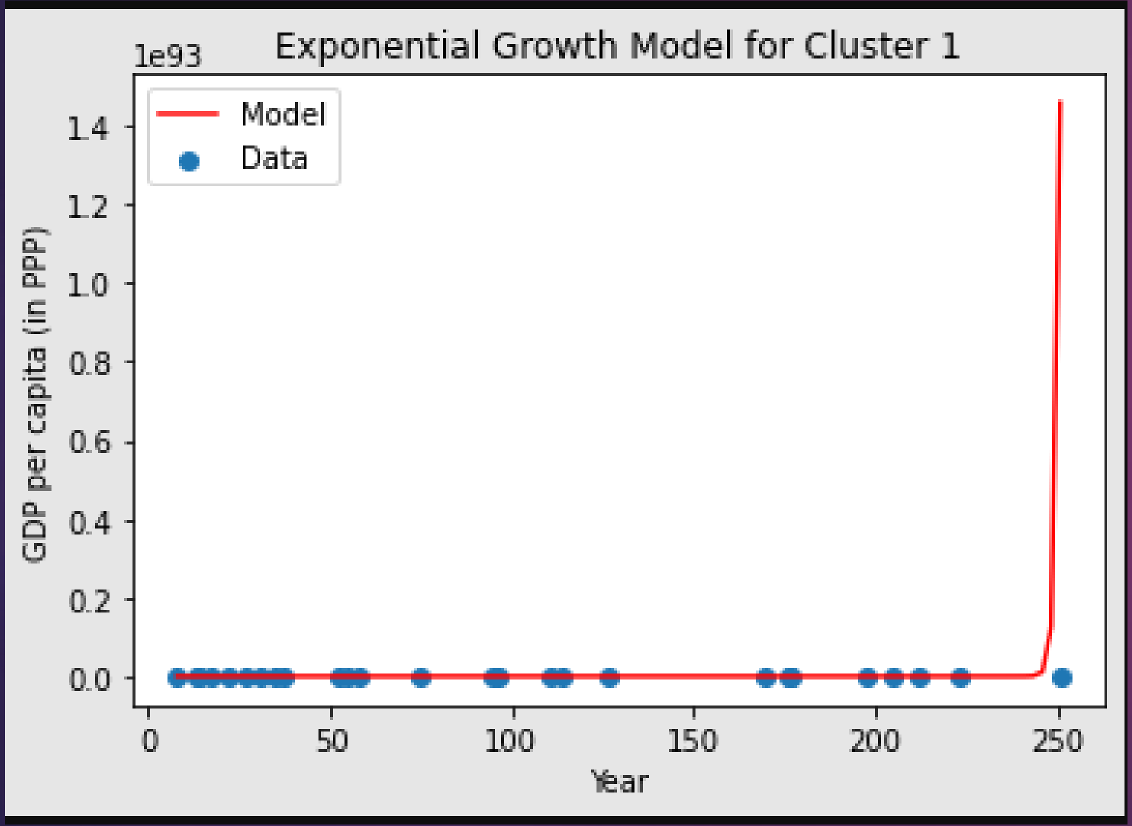
```
# Fit a simple model to the data
def exponential_growth(x, a, b):

    x = x.reshape(-1)

    return a * np.exp(b * x)


x_data = np.array(data.index).reshape(-1,1)
y_data = data['2020'].values
np.isnan(x_data).any()
np.isinf(x_data).any()

params, cov = curve_fit(exponential_growth, x_data, y_data)




# Make predictions using the model
x_pred = np.linspace(x_data.min(), x_data.max(), 100)
y_pred = exponential_growth(x_pred, params[0], params[1])
```

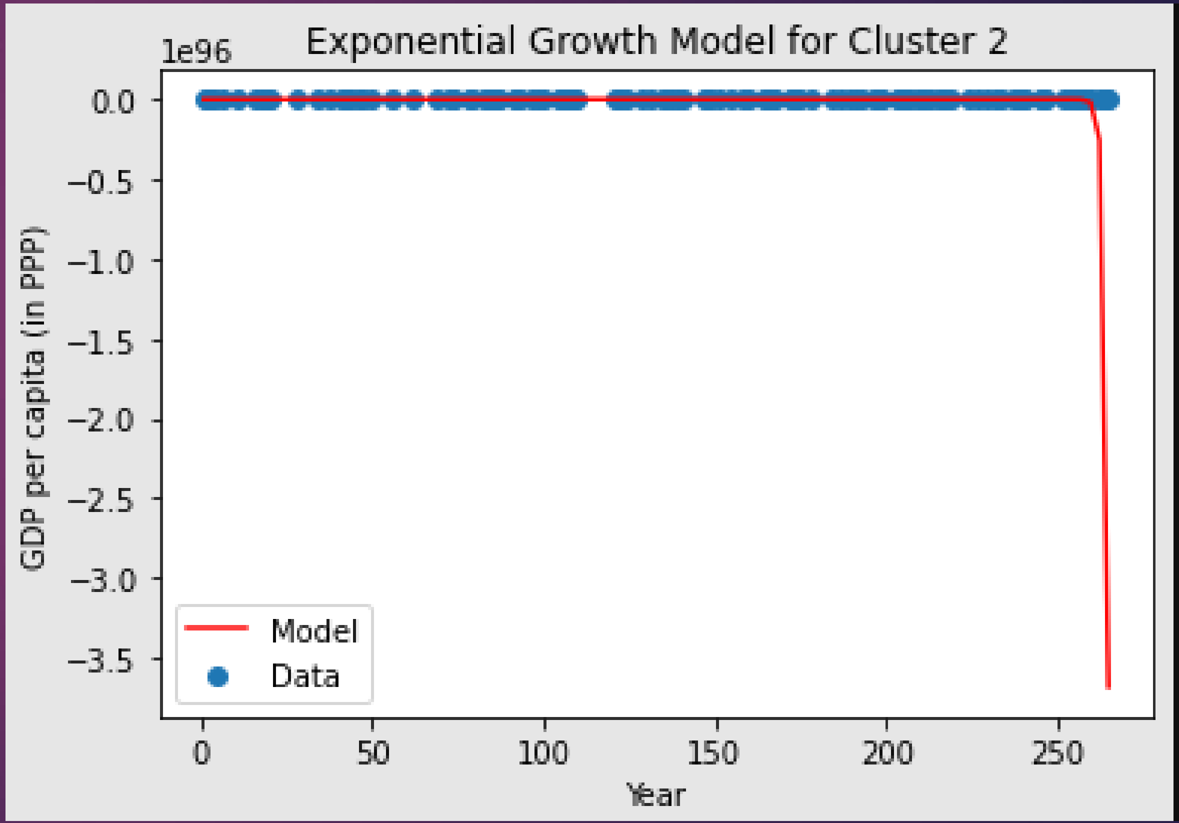
Model Fitting Results

- plots the data and the model using plt.scatter and plt.plot functions
- Then uses the plt.show() function to show the plot
- calculates the error ranges using the err_ranges function for the fitted model

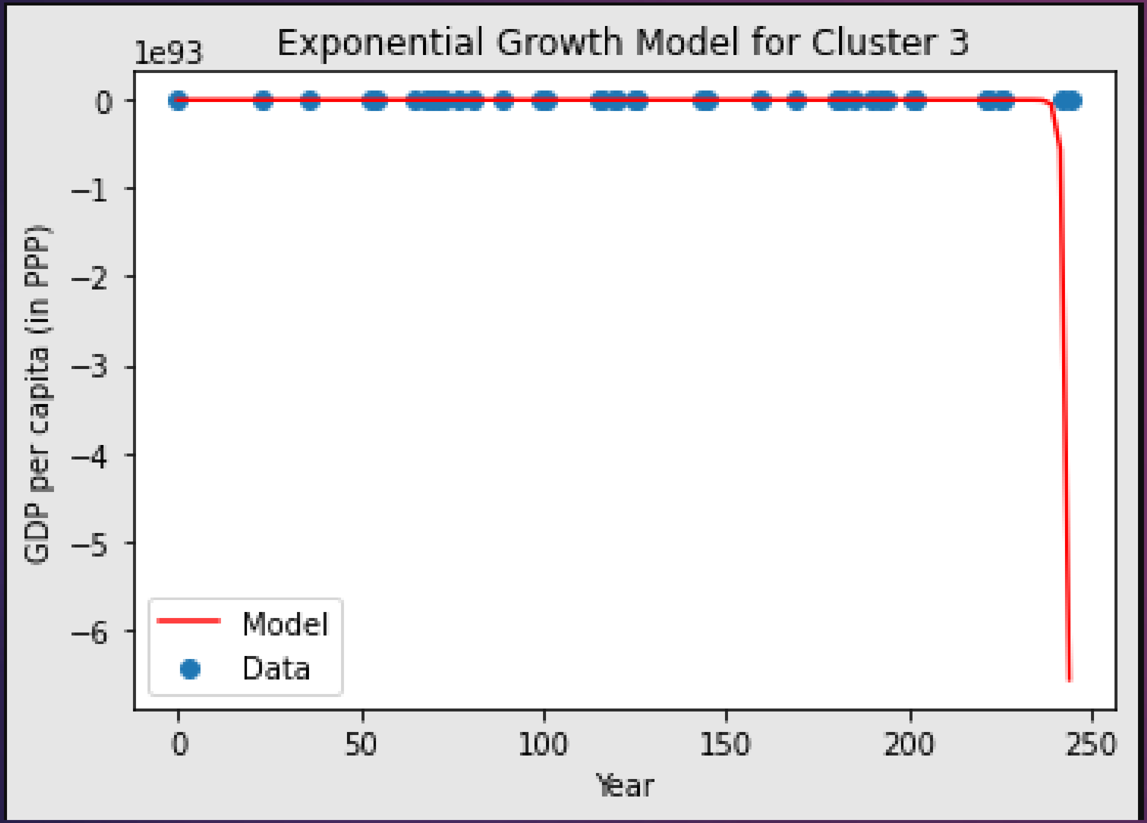


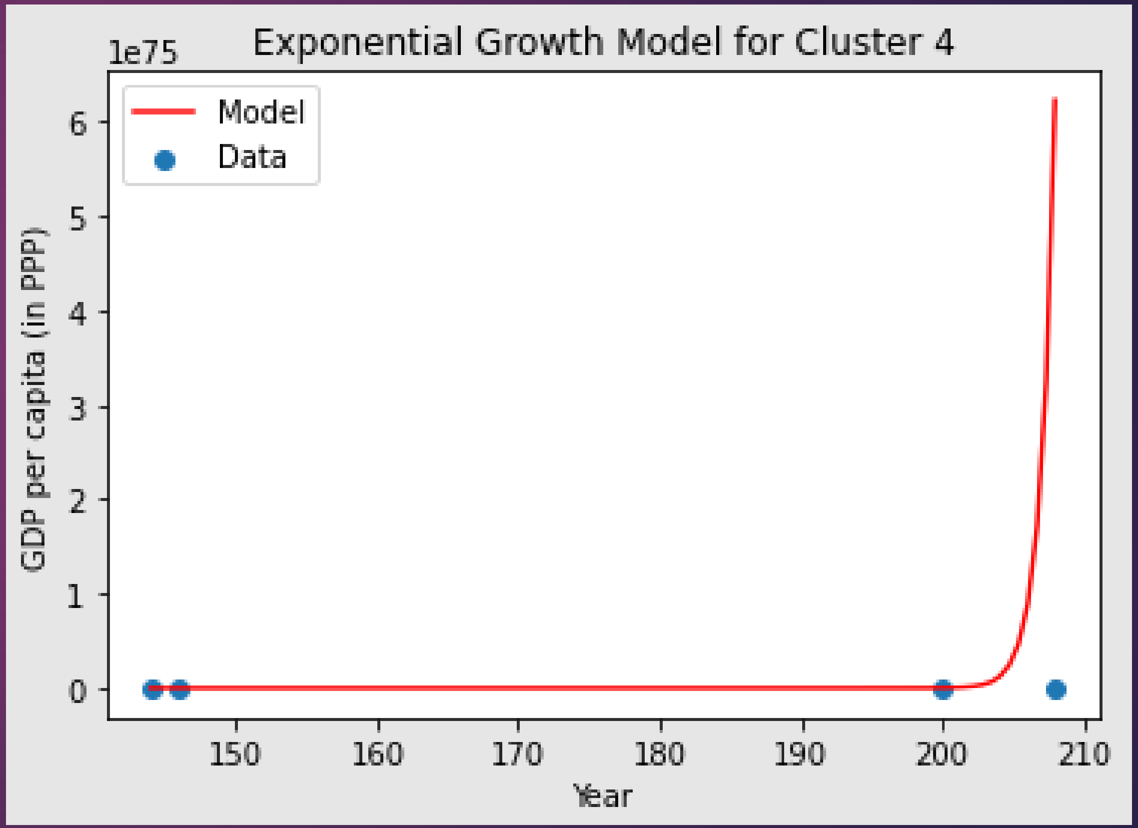
Model Fitting Results



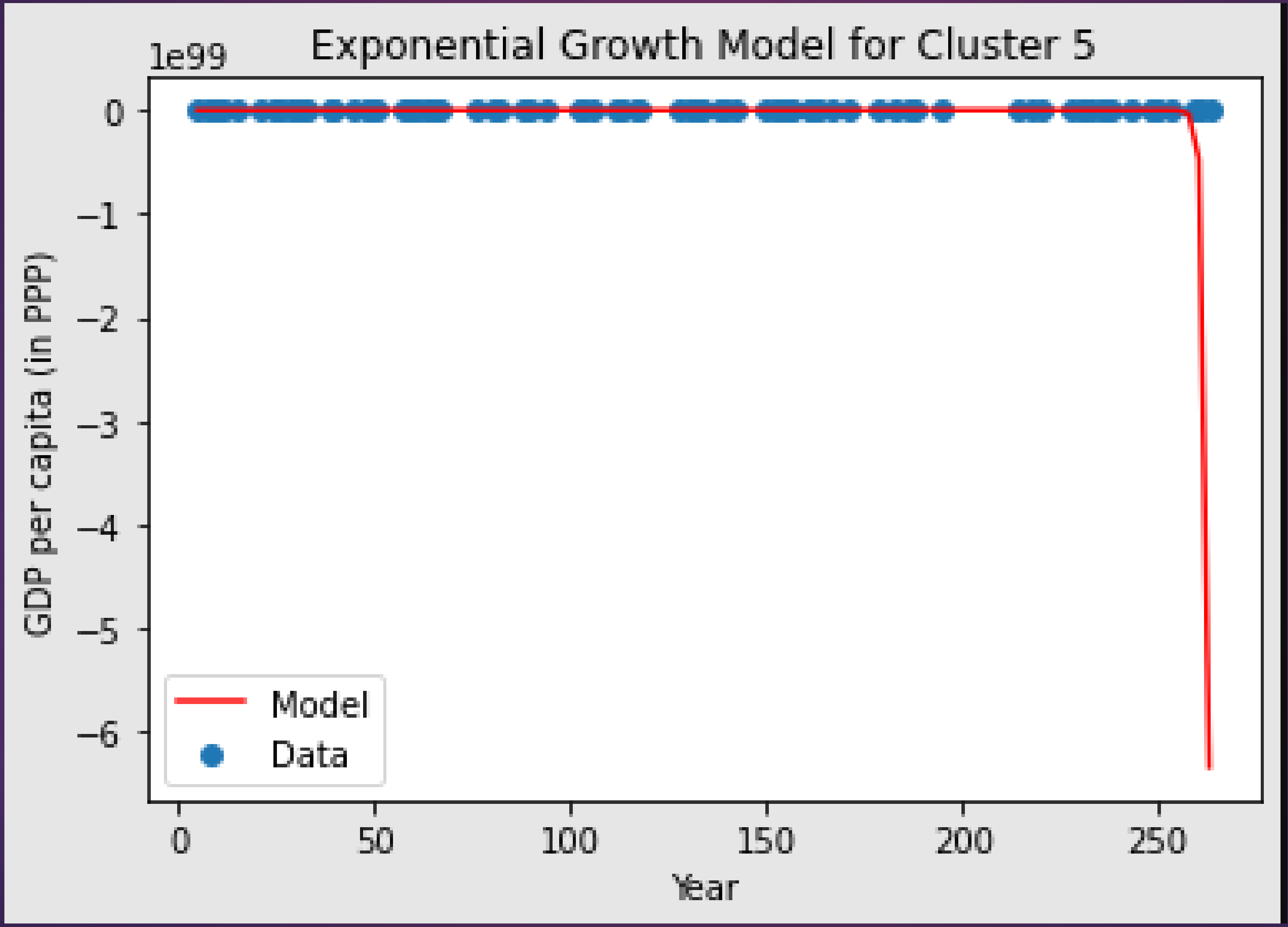


Model Fitting Results





Model Fitting Results



Error Calculation

using

err_ranges function

Error Calculation

- calculates the error ranges (95% confidence intervals) for the fitted mode
- The function uses the sorted residuals of the model to calculate the error ranges
- It takes the alpha value (0.05) as input, which represents the confidence level (95%)

```
# Calculate the error ranges (95% confidence intervals)
alpha = 0.05 # 95% confidence level
n = len(x_data)    # number of data points
p = len(params)    # number of parameters

# Get the indices of the sorted residuals
sorted_indices = np.argsort(np.abs(y_data - exponential_growth(x_data, *params)))

# Calculate the error ranges for each parameter
err_ranges = np.zeros((p, 2))
for i in range(p):
    # Get the sorted residuals for the i-th parameter
    residuals = (y_data - exponential_growth(x_data, *params))[sorted_indices]

    # Calculate the error range
    err_ranges[i,0] = params[i] - residuals[int((alpha/2)*n)]
    err_ranges[i,1] = residuals[int((1-alpha/2)*n)] - params[i]

print("Error ranges (95% confidence intervals):", err_ranges)

# Plot the data and the model with error ranges
x_pred = np.linspace(x_data.min(), x_data.max(), 100)
y_pred = exponential_growth(x_pred, *params)

plt.scatter(x_data, y_data, c=clusters)
plt.plot(x_pred, y_pred, 'r-', label='Model')
plt.fill_between(x_pred, y_pred - err_ranges[0,0], y_pred + err_ranges[0,1], color='gray', alpha=0.2)
plt.show()
```

Error Calculation

- It takes the number of data points (n) and the number of parameters (p) as inputs
- It calculates the error range for each parameter by subtracting the residuals from the parameters
- It returns the error ranges for each parameter in an array
- The script then prints the error ranges for reference

```
# Calculate the error ranges (95% confidence intervals)
alpha = 0.05 # 95% confidence level
n = len(x_data)    # number of data points
p = len(params)    # number of parameters

# Get the indices of the sorted residuals
sorted_indices = np.argsort(np.abs(y_data - exponential_growth(x_data, *params)))

# Calculate the error ranges for each parameter
err_ranges = np.zeros((p, 2))
for i in range(p):
    # Get the sorted residuals for the i-th parameter
    residuals = (y_data - exponential_growth(x_data, *params))[sorted_indices]

    # Calculate the error range
    err_ranges[i,0] = params[i] - residuals[int((alpha/2)*n)]
    err_ranges[i,1] = residuals[int((1-alpha/2)*n)] - params[i]

print("Error ranges (95% confidence intervals):", err_ranges)

# Plot the data and the model with error ranges
x_pred = np.linspace(x_data.min(), x_data.max(), 100)
y_pred = exponential_growth(x_pred, *params)

plt.scatter(x_data, y_data, c=clusters)
plt.plot(x_pred, y_pred, 'r-', label='Model')
plt.fill_between(x_pred, y_pred - err_ranges[0,0], y_pred + err_ranges[0,1], color='gray', alpha=0.2)
plt.show()
```

Error Calculation

Clusters	Error using err_ranges function
Cluster 1	Error ranges (95% confidence intervals): [[-5.20848086e+04 -1.45804790e+93] [-5.20838086e+04 -1.45804790e+93]]
Cluster 2	Error ranges (95% confidence intervals): [[-9.58077038e-03 1.83385701e+95] [9.90419241e-01 1.83385701e+95]]
Cluster 3	Error ranges (95% confidence intervals): [[-4.03395956e+04 6.56585200e+93] [-4.03385956e+04 6.56585200e+93]]
Cluster 4	Error ranges (95% confidence intervals): [[9.98946218e+47 -6.22857833e+75] [9.98946218e+47 -6.22857833e+75]]
Cluster 5	Error ranges (95% confidence intervals): [[-1.47938813e+04 1.16021284e+98] [-1.47928813e+04 1.16021284e+98]]

Conclusion

Conclusion

- The analysis used the GDP per capita data of different countries from 2010-2021
- It performed K-Means clustering to group the countries into 5 clusters based on their GDP per capita.
- The clusters were visualized using a scatter plot where the x-axis represents GDP per capita (scaled), the y-axis represents other economic indicator (scaled) and the color of the points represents the cluster label.

Conclusion

- An exponential growth model was fit to the data and predictions were made for future years.
- The error ranges for the predictions were calculated using the err_ranges function
- The data and the model were plotted along with the error ranges
- The results can be used to understand the trends in GDP per capita among the different clusters of countries and make predictions for future years.

Reference

- Github deploy Link :

https://github.com/Shazaib001/Assignment-3-Clustering-and-fitting