```
# Import libraries, load datasets, displau the first rows of Global Population dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df_GlobPop = pd.read_csv("Unit04 Global_Population.csv")
df_GDP = pd.read_csv("Unit04 Global_GDP.csv")
df_GlobPop.head()
```

	Country Name	Country Code	Series Name	Series Code	1960	1961	1962	1963	1964	1965	...	2012	2013	2(
0	Aruba	ABW	Population, total	SP.POP.TOTL	54211	55438	56225	56695	57032	57360	...	102560	103159	103
1	Afghanistan	AFG	Population, total	SP.POP.TOTL	8996967	9169406	9351442	9543200	9744772	9956318	...	31161378	32269592	33370
2	Africa Eastern and Southern	AFE	Population, total	SP.POP.TOTL	130836765	134159786	137614644	141202036	144920186	148769974	...	547482863	562601578	578075
3	Africa Western and Central	AFW	Population, total	SP.POP.TOTL	96396419	98407221	100506960	102691339	104953470	107289875	...	370243017	380437896	390882
4	Albania	ALB	Population, total	SP.POP.TOTL	1608800	1659800	1711319	1762621	1814135	1864791	...	2900401	2895092	2889

5 rows × 66 columns

```
# Display the first rows of GDP dataset
df_GDP.head()
```

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	...	2(
0	Aruba	ABW	GDP (current US$)	NY.GDP.MKTP.CD	NaN	NaN	NaN	NaN	NaN	NaN	...	2.549721e+
1	Africa Eastern and Southern	AFE	GDP (current US$)	NY.GDP.MKTP.CD	1.929944e+10	1.970954e+10	2.147872e+10	2.571501e+10	2.351080e+10	2.679160e+10	...	9.427110e+
2	Afghanistan	AFG	GDP (current US$)	NY.GDP.MKTP.CD	5.377778e+08	5.488889e+08	5.466667e+08	7.511112e+08	8.000000e+08	1.006667e+09	...	1.780511e+
3	Africa Western and Central	AFW	GDP (current US$)	NY.GDP.MKTP.CD	1.040428e+10	1.112805e+10	1.194335e+10	1.267652e+10	1.383858e+10	1.486247e+10	...	6.709630e+
4	Angola	AGO	GDP (current US$)	NY.GDP.MKTP.CD	NaN	NaN	NaN	NaN	NaN	NaN	...	1.117900e+

5 rows × 65 columns

```
# Create a subset of Global Population dataset for the selected years and display its first rows
df_GlobPop_subset = df_GlobPop[["Country Name"]+[str(year) for year in range(2001,2021)]]
df_GlobPop_subset.head()
```

	Country Name	2001	2002	2003	2004	2005	2006	2007	2008	2009	...	2011	2012	2(
0	Aruba	92898	94992	97017	98737	100031	100834	101222	101358	101455	...	102046	102560	103
1	Afghanistan	21606992	22600774	23680871	24726689	25654274	26433058	27100542	27722281	28394806	...	30117411	31161378	32269
2	Africa Eastern and Southern	408522129	419223717	430246635	441630149	453404076	465581372	478166911	491173160	504604672	...	532760424	547482863	56260
3	Africa Western and Central	274433894	281842480	289469530	297353098	305520588	313985474	322741656	331772330	341050537	...	360285439	370243017	38043
4	Albania	3060173	3051010	3039616	3026939	3011487	2992547	2970017	2947314	2927519	...	2905195	2900401	2895

5 rows × 21 columns

```
# Convert column to numeric ignoring errors
df_GlobPop_subset.iloc[:,2:]=df_GlobPop_subset.iloc[:,2:].apply(pd.to_numeric,errors="coerce")
df_GlobPop_subset = df_GlobPop_subset.dropna()
```

```
<ipython-input-11-623038563e9a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_GlobPop_subset.iloc[:,2:]=df_GlobPop_subset.iloc[:,2:].apply(pd.to_numeric,errors="coerce")
```

```
# Calculate mean of Global Population per country
mean_GlobPop = df_GlobPop_subset.iloc[:,2:].mean(axis=1)
mean_GlobPop_data = pd.DataFrame({"Country Name":df_GlobPop_subset["Country Name"],"Mean_GlobPop":mean_GlobPop})
mean_GlobPop_data.head()
```

	Country Name	Mean_GlobPop
1	Afghanistan	30665934.368421
2	Africa Eastern and Southern	538633817.578947
3	Africa Western and Central	363947065.526316
4	Albania	2929359.631579
5	Algeria	37143227.684211

```
# Create a subset of GDP dataset for the selected years and display its first rows
df_GDP_subset = df_GDP[["Country Name"]+[str(year) for year in range(2001,2021)]]
df_GDP_subset.head()
```

	Country Name	2001	2002	2003	2004	2005	2006	2007	2008	2009	...	
0	Aruba	1.920112e+09	1.941341e+09	2.021229e+09	2.228492e+09	2.330726e+09	2.424581e+09	2.615084e+09	2.745251e+09	2.498883e+09	...	2.5497
1	Africa Eastern and Southern	2.586600e+11	2.647080e+11	3.524430e+11	4.385650e+11	5.118980e+11	5.755690e+11	6.607740e+11	7.078530e+11	7.120540e+11	...	9.427
2	Afghanistan	NaN	4.055180e+09	4.515559e+09	5.226779e+09	6.209138e+09	6.971286e+09	9.747880e+09	1.010931e+10	1.241616e+10	...	1.780
3	Africa Western and Central	1.480120e+11	1.769330e+11	2.046420e+11	2.540900e+11	3.105540e+11	3.932970e+11	4.617780e+11	5.664260e+11	5.069960e+11	...	6.709
4	Angola	8.936064e+09	1.528559e+10	1.781271e+10	2.355205e+10	3.697092e+10	5.238101e+10	6.526645e+10	8.853861e+10	7.030716e+10	...	1.117

5 rows × 21 columns

```
# Convert column to numeric ignoring errors
df_GDP_subset.iloc[:,2:]=df_GDP_subset.iloc[:,2:].apply(pd.to_numeric,errors="coerce")
df_GDP_subset = df_GDP_subset.dropna()
```

```
# Calculate mean of GDP per country
mean_GDP = df_GDP_subset.iloc[:,2:].mean(axis=1)
mean_GDP_data = pd.DataFrame({"Country Name":df_GDP_subset["Country Name"],"Mean_GDP":mean_GDP})
mean_GDP_data.head()
```

	Country Name	Mean_GDP
1	Africa Eastern and Southern	7.697044e+11
3	Africa Western and Central	5.785776e+11
4	Angola	8.235617e+10
5	Albania	1.129206e+10
7	Arab World	2.102552e+12

```
# Calculate mean per Capita GDP per country and remove NaN values
mean_perCapitaGDP = mean_GDP/mean_GlobPop
mean_perCapitaGDP_data = pd.DataFrame({"Country Name":df_GDP_subset["Country Name"],"Mean_perCapitaGDP":mean_perCapitaGDP,"Mean_GlobPop":mean_GlobPop})
mean_perCapitaGDP_data=mean_perCapitaGDP_data.dropna()
mean_perCapitaGDP_data.head()
```

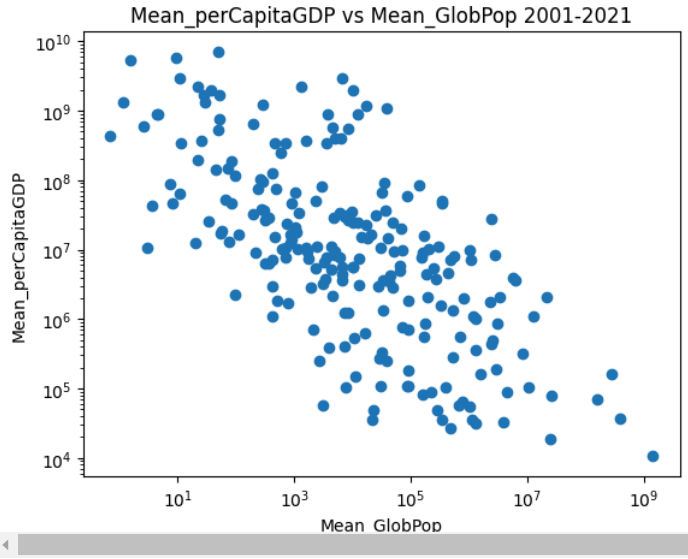
	Country Name	Mean_perCapitaGDP	Mean_GlobPop
1	Africa Eastern and Southern	25099.65657	30665934.368421
3	Africa Western and Central	1589.730173	363947065.526316
4	Angola	28114.05123	2929359.631579
5	Albania	304.014069	37143227.684211
7	Arab World	26549794.871011	79192.789474

```
# Perform correlation
correlation = mean_perCapitaGDP_data['Mean_perCapitaGDP'].corr(mean_perCapitaGDP_data['Mean_GlobPop'])
print(correlation)
```

```
-0.03211364849150776
```

```
# Visualize results
plt.scatter(mean_perCapitaGDP_data["Mean_perCapitaGDP"], mean_perCapitaGDP_data["Mean_GlobPop"])
```

```
plt.title("Mean_perCapitaGDP vs Mean_GlobPop 2001-2021")
plt.xlabel("Mean_GlobPop")
plt.ylabel("Mean_perCapitaGDP")
plt.xscale('log')
plt.yscale('log')
plt.show()
```




```
# Import libraries
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Handle infinite and NaN values
mean_perCapitaGDP_data["Mean_GlobPop"].replace([np.inf, -np.inf], np.nan, inplace=True)
mean_perCapitaGDP_data.dropna(subset=["Mean_GlobPop"], inplace=True)
mean_perCapitaGDP_data["Mean_perCapitaGDP"].replace([np.inf, -np.inf], np.nan, inplace=True)
mean_perCapitaGDP_data.dropna(subset=["Mean_perCapitaGDP"], inplace=True)
```

```
<ipython-input-25-664d3f507dcb>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always b

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)


  mean_perCapitaGDP_data["Mean_GlobPop"].replace([np.inf, -np.inf], np.nan, inplace=True)
<ipython-input-25-664d3f507dcb>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retai
  mean_perCapitaGDP_data["Mean_GlobPop"].replace([np.inf, -np.inf], np.nan, inplace=True)
<ipython-input-25-664d3f507dcb>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always b

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)


  mean_perCapitaGDP_data["Mean_perCapitaGDP"].replace([np.inf, -np.inf], np.nan, inplace=True)
<ipython-input-25-664d3f507dcb>:3: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retai
  mean_perCapitaGDP_data["Mean_perCapitaGDP"].replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
# Define independent (x) and dependent (y) variable
x=np.log(mean_perCapitaGDP_data["Mean_GlobPop"].values.reshape(-1,1))
y=np.log(mean_perCapitaGDP_data["Mean_perCapitaGDP"].values.reshape(-1,1))
```

```
# Split 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)

# Create regression model
regression_model=LinearRegression()

# Train the model
regression_model.fit(x_train,y_train)

# Make predictions on the testing set
y_predict=regression_model.predict(x_test)

# Evaluate the model
mse=mean_squared_error(y_test,y_predict)
r2=r2_score(y_test,y_predict)
slope_model=regression_model.coef_[0][0]
intercept_model=regression_model.intercept_[0]
print(f"Slope: {slope_model}")
print(f"Intercept: {intercept_model}")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
Slope: -0.8635549201381624
Intercept: 22.89072384445634
Mean Squared Error: 8.416878860321656
R-squared: 0.5791055782928793
```

```
# Visualize results
plt.scatter(x_test, y_test, color='blue', label='Actual')
plt.plot(x_test, y_predict, color='red', linewidth=2, label='Predicted')
plt.title("Regression Analysis")
plt.xlabel('Mean_GlobPop')
plt.ylabel('Mean_perCapitaGDP')
plt.legend()
plt.show()
```