```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

(module) pyplot

```
data = pd.read_csv("/content/API_SP.POP.TOTL_DS2_en_csv_v2_38144.csv", skiprows=4)
```

```
data
```

	Code","1960","1961","1962","1963","1964","1965","1966","1967","1968","1969","1970","1971","1972","1973","1974","1975","1976","
0	
1	
2	
3	
4	
...	
261	
262	
263	
264	
265	

266 rows × 1 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.head(200)
```

	Code","1960","1961","1962","1963","1964","1965","1966","1967","1968","1969","1970","1971","1972","1973","1974","1975","1976","
0	
1	
2	
3	
4	
...	
195	
196	
197	
198	
199	

200 rows × 1 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
# prompt: Using dataframe data: Select the Variable

print(data.columns)
# Update the column selection after verifying the correct column names from the print output
data[['Country Name',"Country Code","Indicator Name","Indicator Code","1960","1961","1962","1963","1964","1965","1966","1967","1968","19
```

```
Index(['Country Name', 'Country Code', 'Indicator Name', 'Indicator Code',
       '1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967', '1968',
       '1969', '1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977',
       '1978', '1979', '1980', '1981', '1982', '1983', '1984', '1985', '1986',
       '1987', '1988', '1989', '1990', '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', '2021', '2022',
       '2023', '2024', 'Unnamed: 69'],
      dtype='object')
```

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	...	
0	Aruba	ABW	Population, total	SP.POP.TOTL	54922.0	55578.0	56320.0	57002.0	57619.0	58190.0	...	
1	Africa Eastern and Southern	AFE	Population, total	SP.POP.TOTL	130075728.0	133534923.0	137171659.0	140945536.0	144904094.0	149033472.0	...	60
2	Afghanistan	AFG	Population, total	SP.POP.TOTL	9035043.0	9214083.0	9404406.0	9604487.0	9814318.0	10036008.0	...	3
3	Africa Western and Central	AFW	Population, total	SP.POP.TOTL	97630925.0	99706674.0	101854756.0	104089175.0	106388440.0	108772632.0	...	41
4	Angola	AGO	Population, total	SP.POP.TOTL	5231654.0	5301583.0	5354310.0	5408320.0	5464187.0	5521981.0	...	2
...	...	...	...	...	...	...	...	...	...	...	...	
261	Kosovo	XKX	Population, total	SP.POP.TOTL	984846.0	1011421.0	1036950.0	1062737.0	1090270.0	1120168.0	...	
262	Yemen, Rep.	YEM	Population, total	SP.POP.TOTL	5532301.0	5655232.0	5782221.0	5911135.0	6048006.0	6195593.0	...	3
263	South Africa	ZAF	Population, total	SP.POP.TOTL	16440172.0	16908035.0	17418522.0	17954564.0	18511361.0	19089380.0	...	5
264	Zambia	ZMB	Population, total	SP.POP.TOTL	3153729.0	3254086.0	3358099.0	3465907.0	3577017.0	3692086.0	...	1
265	Zimbabwe	ZWE	Population, total	SP.POP.TOTL	3809389.0	3930401.0	4055959.0	4185877.0	4320006.0	4458462.0	...	1

266 rows × 69 columns

```
# prompt: some procedure that not affect the data set

# This code block only displays selected columns and does not modify the original 'data' DataFrame.
# You can perform various operations on this selection, such as printing or further filtering,
# without affecting the content of 'data'.
selected_data = data[['Country Name',"Country Code","Indicator Name","Indicator Code","1960","1961","1962","1963","1964","1965","1966",'

# Example of a procedure that does not affect the original data:
# Printing the shape of the selected data
print("Shape of selected data:", selected_data.shape)

# Example of another procedure that does not affect the original data:
# Calculating the mean of a specific year's column in the selected data
# Make sure the column exists and contains numeric data before doing this
try:
  mean_1960 = selected_data['1960'].mean()
  print("Mean of 1960 population in selected data:", mean_1960)
except KeyError:
  print("Column '1960' not found in selected data or is not numeric.")
except Exception as e:
  print(f"An error occurred: {e}")

# You can add more procedures here that only operate on 'selected_data'
# and do not assign the result back to 'data'.
```

```
Shape of selected data: (266, 69)
Mean of 1960 population in selected data: 115402287.79545455
```

```
# prompt: is there any null values

print(data.isnull().sum())
```

```
Country Name        0
Country Code        0
Indicator Name      0
Indicator Code      0
```

```
1960                2
                  ...
2021                1
2022                1
2023                1
2024                1
Unnamed: 69       266
Length: 70, dtype: int64
```

(module) pyplot

```
# prompt: by adding some missing values into the mode, medium and sequence of non missing terms in the data det

# Select a column that likely contains numerical data for mode and median calculation
# Let's assume '2022' is a suitable column from the selected_data
numeric_column = '2022'

# Check if the column exists and is numeric
if numeric_column in selected_data.columns:
  # Calculate the mode
  try:
    # mode() can return multiple values if there's a tie
    data_mode = selected_data[numeric_column].mode()
    print(f"\nMode of {numeric_column}:\n{data_mode}")
  except Exception as e:
    print(f"\nCould not calculate mode for {numeric_column}: {e}")

  # Calculate the median
  try:
    data_median = selected_data[numeric_column].median()
    print(f"\nMedian of {numeric_column}: {data_median}")
  except Exception as e:
    print(f"\nCould not calculate median for {numeric_column}: {e}")

  # Find the sequence of non-missing terms
  try:
    # Get the sequence of non-missing values
    non_missing_sequence = selected_data[numeric_column].dropna().tolist()
    print(f"\nSequence of non-missing terms in {numeric_column}:")
    # Print only the first few elements if the sequence is very long
    if len(non_missing_sequence) > 100:
        print(non_missing_sequence[:100], "...")
    else:
        print(non_missing_sequence)
  except Exception as e:
    print(f"\nCould not get sequence of non-missing terms for {numeric_column}: {e}")

else:
  print(f"\nColumn '{numeric_column}' not found in the selected data. Please choose a valid column.")

# Note: The task asks about adding missing values into the mode, median, and sequence.
# However, mode and median are measures calculated *from* the data, and the sequence
# is the sequence of existing non-missing values. You don't "add missing values into"
# these concepts directly. The presence of missing values affects their calculation.
# The code above calculates these measures and the non-missing sequence using the data,
# taking into account any existing missing values. If you intended to *impute* missing
# values before calculating these, that would require a different approach (e.g., using fillna).
# The current code calculates these based on the data *as is*.
```

```
Mode of 2022:
0    1.229209e+09
1    1.648010e+09
Name: 2022, dtype: float64

Median of 2022: 10486941.0

Sequence of non-missing terms in 2022:
[107310.0, 731821393.0, 40578842.0, 497387180.0, 35635029.0, 2777689.0, 79705.0, 471352066.0, 10074977.0, 45407904.0, 2969200.0, 48
```

```
data
```

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	...	
0	Aruba	ABW	Population, total	SP.POP.TOTL	54922.0	55578.0	56320.0	57002.0	57619.0	58190.0	...	
						(module) pyplot						
1	Africa Eastern and Southern	AFE	Population, total	SP.POP.TOTL	130075728.0	133534923.0	137171659.0	140945536.0	144904094.0	149033472.0	...	62
2	Afghanistan	AFG	Population, total	SP.POP.TOTL	9035043.0	9214083.0	9404406.0	9604487.0	9814318.0	10036008.0	...	3
3	Africa Western and Central	AFW	Population, total	SP.POP.TOTL	97630925.0	99706674.0	101854756.0	104089175.0	106388440.0	108772632.0	...	42
4	Angola	AGO	Population, total	SP.POP.TOTL	5231654.0	5301583.0	5354310.0	5408320.0	5464187.0	5521981.0	...	2
...	...	...	...	...	...	...	...	...	...	...	...	
261	Kosovo	XKX	Population, total	SP.POP.TOTL	984846.0	1011421.0	1036950.0	1062737.0	1090270.0	1120168.0	...	
262	Yemen, Rep.	YEM	Population, total	SP.POP.TOTL	5532301.0	5655232.0	5782221.0	5911135.0	6048006.0	6195593.0	...	3
263	South Africa	ZAF	Population, total	SP.POP.TOTL	16440172.0	16908035.0	17418522.0	17954564.0	18511361.0	19089380.0	...	5
264	Zambia	ZMB	Population, total	SP.POP.TOTL	3153729.0	3254086.0	3358099.0	3465907.0	3577017.0	3692086.0	...	1
265	Zimbabwe	ZWE	Population, total	SP.POP.TOTL	3809389.0	3930401.0	4055959.0	4185877.0	4320006.0	4458462.0	...	1

266 rows × 70 columns

```
print(data.isnull().sum())
```

```
Country Name        0
Country Code        0
Indicator Name      0
Indicator Code      0
1960                2
                  ...
2021                1
2022                1
2023                1
2024                1
Unnamed: 69       266
Length: 70, dtype: int64
```

```
# prompt: 2021,2022,2023,2024 is there will be a missing term available so make it has non missing terms

# To see if there are missing values in the years 2021, 2022, 2023, 2024
print("\nMissing values in recent years:")
print(data[['2021', '2022', '2023', '2024']].isnull().sum())

# The prompt mentions making terms "non missing". This usually implies imputation
# For this example, let's fill missing values in the recent years with the median
# of that specific year's column.
years_to_impute = ['2021', '2022', '2023', '2024']

for year in years_to_impute:
  if year in data.columns:
    # Calculate the median for the current year, excluding existing NaNs
    median_value = data[year].median()
    # Fill NaN values in the current year with the calculated median
    data[year] = data[year].fillna(median_value)
    print(f"\nFilled missing values in column '{year}' with median ({median_value:.2f}).")
  else:
    print(f"\nColumn '{year}' not found in the data.")

# Verify that the missing values have been filled in the specified columns
print("\nMissing values after imputation:")
print(data[['2021', '2022', '2023', '2024']].isnull().sum())

# Now, recalculate the mode, median, and sequence of non-missing terms for '2022'
# using the DataFrame after imputation.
numeric_column_after_imputation = '2022'

if numeric_column_after_imputation in data.columns:
```

```
    # Calculate the mode after imputation
    try:
      data_mode_imputed = data[numeric_column_after_imputation].mode()
      print(f"\nMode of {numeric_column_after_imputation} after imputation:\n{data_mode_imputed}")
    except Exception as e:
      print(f"\nCould not calculate mode for {numeric_column_after_imputation} after imputation: {e}")
                                                                    (module) pyplot
    # Calculate the median after imputation
    try:
      data_median_imputed = data[numeric_column_after_imputation].median()
      print(f"\nMedian of {numeric_column_after_imputation} after imputation: {data_median_imputed}")
    except Exception as e:
      print(f"\nCould not calculate median for {numeric_column_after_imputation} after imputation: {e}")

    # Find the sequence of non-missing terms after imputation
    try:
      # After fillna, dropna() should return the entire column (if no NaNs were originally present,
      # or all NaNs were filled).
      non_missing_sequence_imputed = data[numeric_column_after_imputation].dropna().tolist()
      print(f"\nSequence of non-missing terms in {numeric_column_after_imputation} after imputation:")
      if len(non_missing_sequence_imputed) > 100:
          print(non_missing_sequence_imputed[:100], "...")
      else:
          print(non_missing_sequence_imputed)
    except Exception as e:
      print(f"\nCould not get sequence of non-missing terms for {numeric_column_after_imputation} after imputation: {e}")

  else:
    print(f"\nColumn '{numeric_column_after_imputation}' not found in the data after potential imputation.")

  # Display the updated data head to see the filled values
  print("\nData head after imputation:")
  print(data[['Country Name', '2021', '2022', '2023', '2024']].head())
```

```
Missing values in recent years:
2021    1
2022    1
2023    1
2024    1
dtype: int64

Filled missing values in column '2021' with median (10505772.00).

Filled missing values in column '2022' with median (10486941.00).

Filled missing values in column '2023' with median (10644851.00).

Filled missing values in column '2024' with median (10876981.00).

Missing values after imputation:
2021    0
2022    0
2023    0
2024    0
dtype: int64

Mode of 2022 after imputation:
0    1.048694e+07
1    1.229209e+09
2    1.648010e+09
Name: 2022, dtype: float64

Median of 2022 after imputation: 10486941.0

Sequence of non-missing terms in 2022 after imputation:
[107310.0, 731821393.0, 40578842.0, 497387180.0, 35635029.0, 2777689.0, 79705.0, 471352066.0, 10074977.0, 45407904.0, 2969200.0, 48

Data head after imputation:
                   Country Name         2021         2022         2023  \
0                         Aruba     107700.0     107310.0     107359.0
1  Africa Eastern and Southern  713090928.0  731821393.0  750503764.0
2                   Afghanistan   40000412.0   40578842.0   41454761.0
3   Africa Western and Central  485920997.0  497387180.0  509398589.0
4                        Angola   34532429.0   35635029.0   36749906.0

          2024
0     107624.0
1  769294618.0
2   42647492.0
3  521764076.0
4   37885849.0
```

```
data
```

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	...	
0	Aruba	ABW	Population, total	SP.POP.TOTL	54922.0	55578.0	56320.0	57002.0	57619.0	58190.0	...	
						(module) pyplot						
1	Africa Eastern and Southern	AFE	Population, total	SP.POP.TOTL	130075728.0	133534923.0	137171659.0	140945536.0	144904094.0	149033472.0	...	62
2	Afghanistan	AFG	Population, total	SP.POP.TOTL	9035043.0	9214083.0	9404406.0	9604487.0	9814318.0	10036008.0	...	3
3	Africa Western and Central	AFW	Population, total	SP.POP.TOTL	97630925.0	99706674.0	101854756.0	104089175.0	106388440.0	108772632.0	...	42
4	Angola	AGO	Population, total	SP.POP.TOTL	5231654.0	5301583.0	5354310.0	5408320.0	5464187.0	5521981.0	...	2
...	...	...	...	...	...	...	...	...	...	...	...	
261	Kosovo	XKX	Population, total	SP.POP.TOTL	984846.0	1011421.0	1036950.0	1062737.0	1090270.0	1120168.0	...	
262	Yemen, Rep.	YEM	Population, total	SP.POP.TOTL	5532301.0	5655232.0	5782221.0	5911135.0	6048006.0	6195593.0	...	3
263	South Africa	ZAF	Population, total	SP.POP.TOTL	16440172.0	16908035.0	17418522.0	17954564.0	18511361.0	19089380.0	...	5
264	Zambia	ZMB	Population, total	SP.POP.TOTL	3153729.0	3254086.0	3358099.0	3465907.0	3577017.0	3692086.0	...	1
265	Zimbabwe	ZWE	Population, total	SP.POP.TOTL	3809389.0	3930401.0	4055959.0	4185877.0	4320006.0	4458462.0	...	1

266 rows × 70 columns

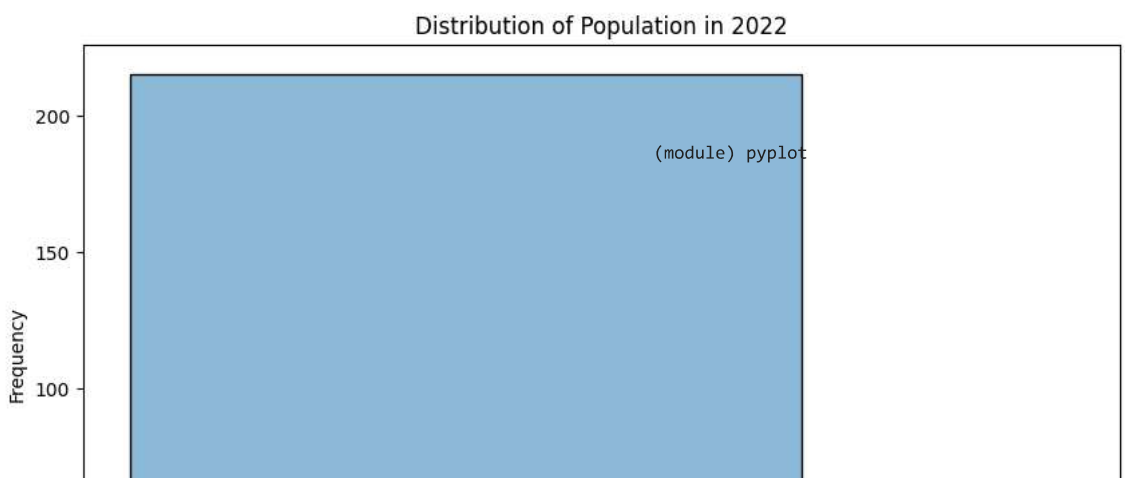
```
print(data.isnull().sum())
```

```
Country Name        0
Country Code        0
Indicator Name      0
Indicator Code      0
1960                2
                  ...
2021                0
2022                0
2023                0
2024                0
Unnamed: 69       266
Length: 70, dtype: int64
```

Generate | Histogram for age | Close

1 of 1 | Undo Changes | Use code with caution

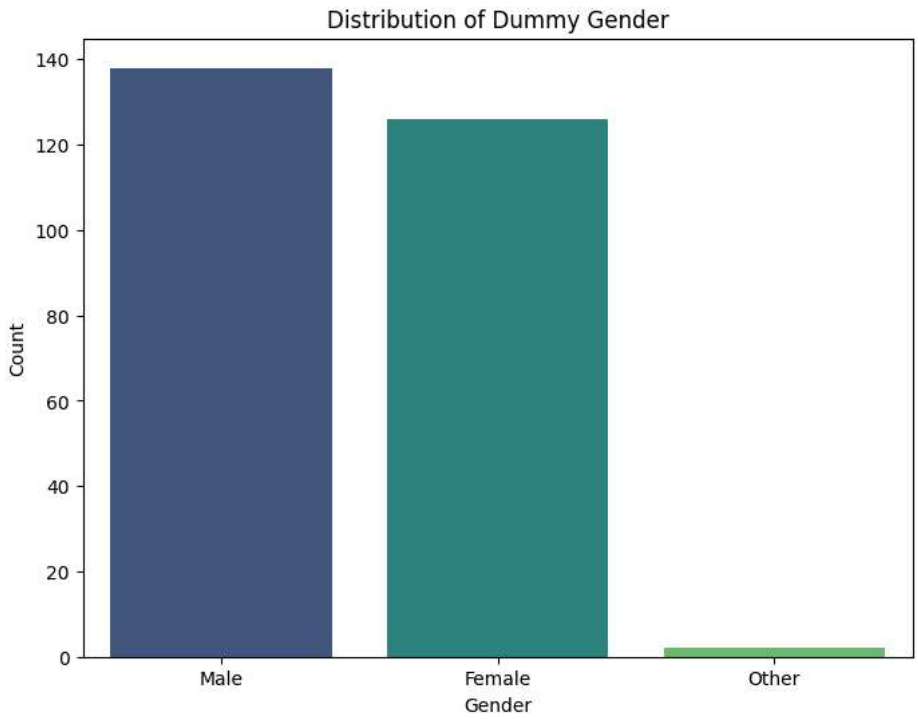
```
import matplotlib.pyplot as plt
population_column = '2022'

if population_column in data.columns:
  plt.figure(figsize=(10, 6))
  sns.histplot(data=data, x=population_column, bins=50, kde=True)
  plt.title(f'Distribution of Population in {population_column}')
  plt.xlabel('Population')
  plt.ylabel('Frequency')
  plt.xscale('log')
  plt.show()
else:
  print(f"The DataFrame does not have a '{population_column}' column.")
```




```
import matplotlib.pyplot as plt
```

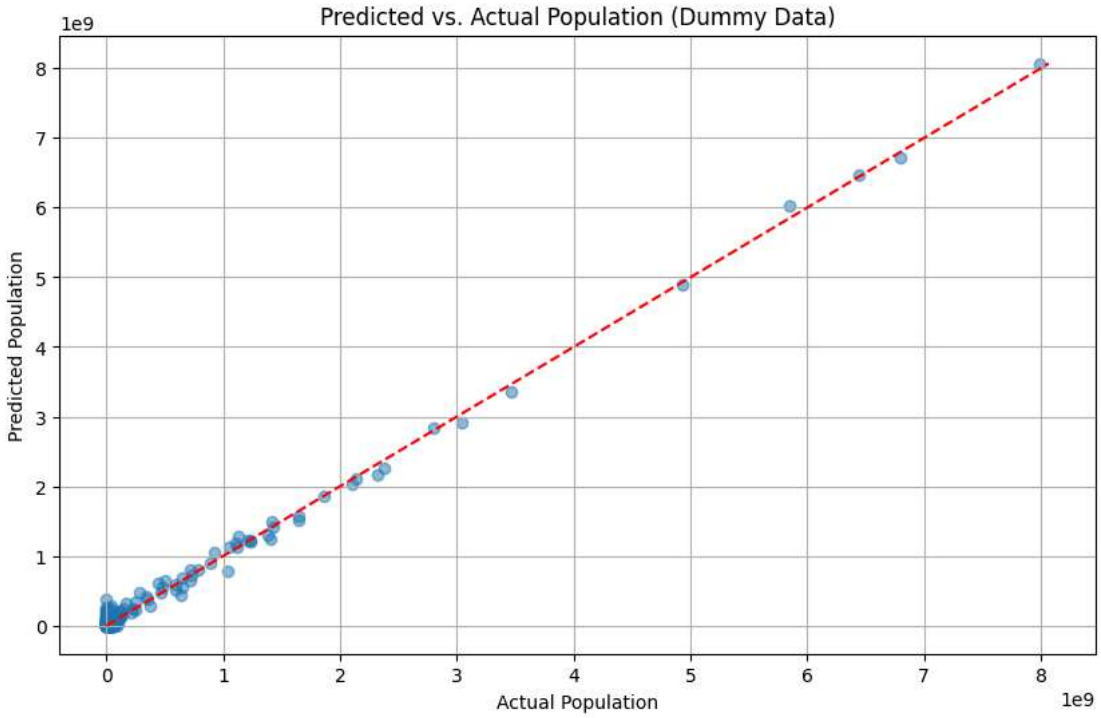
```
import matplotlib.pyplot as plt
import numpy as np

dummy_genders = np.random.choice(['Male', 'Female', 'Other'], size=len(data), p=[0.48, 0.51, 0.01])
data['gender'] = dummy_genders

plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='gender', palette='viridis')
plt.title('Distribution of Dummy Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

```
/tmp/ipython-input-26-211475851.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(data=data, x='gender', palette='viridis')
```




```
import matplotlib.pyplot as plt
import numpy as np

if '2022' in data.columns:
  y_true = data['2022'].dropna().values
  np.random.seed(42)
  noise = np.random.normal(0, y_true.std() * 0.1, y_true.shape)
  y_pred = y_true + noise
  y_pred[y_pred < 0] = 0
```

```
  plt.figure(figsize=(10, 6))
  plt.scatter(y_true, y_pred, alpha=0.5)
  plt.title('Predicted vs. Actual Population (Dummy Data)')
  plt.xlabel('Actual Population')
  plt.ylabel('Predicted Population')
  plt.grid(True)
                                                                    (module) pyplot

  max_val = max(y_true.max(), y_pred.max())
  plt.plot([0, max_val], [0, max_val], 'r--')
  plt.show()

else:
  print("The '2022' column is not available to create dummy true/predicted values for plotting.")
```




```
from google.colab import files
data.to_csv('population_data.csv', index=False)
files.download('population_data.csv')
```