In [4]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Lab 2 : Web scraping and API requests

In this lab exercise you will practice scraping data from a website, as well as doing some priliminary analysis on them.

Deadline: Sunday, Oct 10 11:59

Part 1: Scraping Data From Wikipedia

We have completed a similar task during lecture. You have to scrap a specific page of Wikipedia and answer some questions regarding the data you have collected. You have to get the data about different countries and their respective populations from the following page: https://en.wikipedia.org/wiki/List_of_countries_by_past_and_future_population (https://en.wikipedia.org/wiki/List_of_countries_by_past_and_future_population)

This page contains multiple tables for past and future population of countries. For the first part of this lab do the following:

1. Fetch the data from wikipedia with "requests" library
2. Parse html data with BeautifulSoup library
3. Use BeautifulSoup to extract specific tables
4. Combine the tables and convert the data into a dictionary
5. Make a pandas dataframe from the dictionary
6. Answer some questions and do some basic visualization!

1.1 Get the data from wikipedia (5 pts)

Use "requests" library.

In [5]:

```
# Your code here
import requests
from bs4 import BeautifulSoup

url =requests.get('https://en.wikipedia.org/wiki/List_of_countries_by_past_and_future_popul
```

1.2 Parse html data with BeautifulSoup

Parse the data using BeautifulSoup. Remember that BeautifulSoup has many useful attributes such as prettify(), find(attribute), and find_all(attribute). Check the documentation for more info: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

(https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

1.2.a Find the first title object and extract and print the string stored in it (5 pts)

In [6]:

```
soup = BeautifulSoup(url, 'html.parser')
f_title=soup.find('title').string
print(f_title)
```

```
List of countries by past and projected future population - Wikipedia
```

1.2.b Find all the paragrpahs, store them in a list, and print the first 10 (5 pts)

In [7]:

```
# Your code here
par = soup.find_all("p")
f_ten = []
for i in range (10):
    f_ten.append(par[i].text)
print(f_ten)
```

```
['All the figures shown here have been sourced from the International Data B
ase (IDB) Division of the United States Census Bureau. Every individual valu
e has been rounded to the nearest thousand, to assure data coherence, partic
ularly when adding up (sub)totals. Although data from specific statistical o
ffices may be more accurate, the information provided here has the advantage
of being homogeneous.\n', 'Population estimates, as long as they are based o
n recent censuses, can be more easily projected into the near future than ma
ny macroeconomic indicators, such as GDP, which are much more sensitive to p
olitical and/or economic crises. This means that demographic estimates for t
he next five (or even ten) years can be more accurate than the projected evo
lution of GDP over the same time period (which may also be distorted by infl
ation).\n', 'However, no projected population figures can be considered exac
t. As the IDB states, "figures beyond the years 2020-2025 should be taken wi
th caution", as the "census way towards those years has yet to be paved". Th
us projections can be said to be looking through a kind of "cloudy glass"[1]
or a "misty window": realistically, the projections are "guesstimates".\n',
'To make things complicated, not all countries carry out censuses regularly,
especially some of the poorer, faster-growing sub-Saharan African nations (w
hose evolution may be more interesting, from a demographer\'s point of view,
than the "stagnated" populations of countries like Germany or Italy). As is
well known from the statistics, the population of many sub-Saharan nations,
as well as other nations like Iraq, Egypt and Pakistan, with their low level
of family planning, are growing much faster than in the aging European natio
ns or Japan. \n', "On the other hand, some other countries, like the small A
sian state of Bhutan, have only recently had a thorough census for the first
time: In Bhutan's case in particular, before its national 2005 population su
rvey,[2][3][4] the IDB estimated its population at over 2 million; this was
drastically reduced when the new census results were finally included in its
database.\n", 'Besides, the IDB usually takes some time before including new
data, as happened in the case of Indonesia. That country was reported by the
IDB to have an inflated population of some 242 million by mid-2005, because
it had not still processed the final results of the 2000 Indonesian census.
[5][6][7][8] There was a similar discrepancy with the relatively recent Ethi
opian 2007 census,[9][10] which gave a preliminary result of "only" 73,918,5
05 inhabitants.\n', 'The largest absolute potential discrepancies are natura
lly related to the most populous nations. However, smaller states, such as T
uvalu, can have large relative discrepancies. For instance, the 2002 census
in that Oceanian island, which gave a final population of 9,561[11] shows th
at IDB estimates can be significantly off.\n', 'The national 1 July, mid-yea
r population estimates (usually based on past national censuses) supplied in
these tables are given in thousands.\n', 'The table columns can be sorted by
clicking on their respective heading.\n', 'The retrospective figures use the
present-day names and world political division: for example, the table gives
data for each of the 15 republics of the former Soviet Union, as if they had
already been independent in 1950. The opposite is the case for Germany, whic
h had been divided since the end of the Second World War but was reunified o
n October 3, 1990.\n']
```

1.3 Extract the tables (10 pts)

1.c Extract the tables (10 pts)

We only care about the tables that contain historical population data. Extract all of them.

In [8]:

```
# Your code here
# You need to  find all objects that include the css class “wikitable” within the soup obje

tables  = soup.find_all(class_="wikitable")

print(tables[0].prettify())
```

```
<table class="sortable wikitable" style="text-align: right">
 <tbody>
  <tr>
   <th>
    Country (or dependent territory)
   </th>
   <th>
    1950
   </th>
   <th>
    1955
   </th>
   <th>
    %
   </th>
   <th>
    1960
   </th>
   <th>
    %
```

In [9]:

```
# check the tables you extracted

from IPython.core.display import display, HTML
display(HTML(tables[0].prettify()))
```

Antigua and Barbuda (/wiki/Antigua_and_Barbuda)	46	52	2.19	55	1.32	60
Argentina (/wiki/Argentina)	17,151	18,928	1.99	20,617	1.72	22,284
Armenia (/wiki/Armenia)	1,356	1,566	2.92	1,869	3.61	2,206
Aruba (/wiki/Aruba)	50	54	1.62	58	1.21	60
Australia (/wiki/Australia)	8,268	9,278	2.33	10,362	2.24	11,440
Austria (/wiki/Austria)	6,936	6,947	0.03	7,048	0.29	7,271
Azerbaijan (/wiki/Azerbaijan)	2,886	3,314	2.81	3,882	3.21	4,567
Bahamas (/wiki/The_Bahamas)	71	88	4.33	113	5.19	140

1.4 Convert the tables into a dictionary (35 pts)

Looking at the tables, we only care about the population number throughout the history. You want to associate each country with a series of population values to make a proper time series table you can use to analyze the population in a given coutnry.

First, you need to clean the tables cells from any footnote, links, commas or any garbage values. Once your data is cleaned, make a dictionary and combine each country with its corresponding year/population values across all three tables. An entry in your final dictionary should look like this:

'Albania': {'1950': 1228, '1955': 1393, '1960': 1624, '1965': 1884, '1970': 2157, '1975': 2402, '1980': 2672, '1985': 2957, '1990': 3245, '1995': 3159, '2000': 3159, '2005': 3025, '2010': 2987, '2015': 3030, '2020': 3075, '2025': 3105, '2030': 3103, '2035': 3063, '2040': 2994, '2045': 2913, '2050': 2825},

One way to do it is:

1. First extract the header
2. From your header only store values that are numeric (you can use isnumeric() function, recall that we only care about year values and we don't want to store columns represented by %
3. Once you have all the relevant column names (column that correspond to a year value), you can go over every row of the table
   - Create a dictionary key with the country name
   - Collect and add values corresponding to one of your column names to the dictionary

In [19]:

```
# Your code here
headers = soup.find_all('th')
c_headers = []
for i in range(len(headers)):
    if headers[i].text.isnumeric():
        c_headers.append(headers[i].text)

# Your code here
headers = soup.find_all('th')
c_headers = []
for i in range(len(headers)):
    if headers[i].text.isnumeric():
        c_headers.append(headers[i].text)

#data list
all_data = {}
inside_data = []
final_country_pop = []

#iterate thru every table

# to keep track of whether all the neccessary indices has been created from first table
# to prevent accessing wrong index
iterr = 0
roww = 0
for table in tables:
    for row in table.find_all('tr')[1:]:
        c = [data for data in row.find_all('td')]
        c_pop1 = [x.string for x in [data for data in row.find_all('td')][1:]]
        c_pop1 = [x.replace(",","") for x in c_pop1 if  x!= None]
        c_pop1 = [x for x in c_pop1 if x.isdecimal() == True]
        if (iterr >= 1):
            final_country_pop[roww].extend(c_pop1)
        else:
            final_country_pop.append(c_pop1)
        all_data[c[0].find('a').string] = {}
        keys = list(all_data.keys())
        roww = roww + 1
    iterr = iterr + 1
    roww = 0

final_country_pop = final_country_pop

count = 0
for keys in list(all_data.keys()):
    for headers in range(len(c_headers)):

        all_data[keys][c_headers[headers]] = final_country_pop[count][headers]
    count = count + 1
del all_data['World']
all_data
```

Out[19]:

```
{'Afghanistan': {'1950': '8151',
  '1955': '8892',
  '1960': '9830',
  '1965': '10998',
  '1970': '12431',
  '1975': '14133',
  '1980': '15045',
  '1985': '13120',
  '1990': '13569',
  '1995': '19446',
  '2000': '22462',
  '2005': '26335',
  '2010': '29121',
  '2015': '32565',
  '2020': '36644',
  '2025': '41118',
  '2030': '45665',
```

1.5 Create a dataframe from your dictionary (10 pts)

Now that all tables are stored in a dictionary, we can convert the dictionary into a pandas dataframe.

1. Remove the "World" row
2. Replace 'NaN' values with 0
3. Display the first 8 rows

In [11]:

```
import pandas as pd
df =pd.DataFrame(all_data)
df.head(8)
```

Out[11]:

	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Anguilla	Antigua and Barbuda	Argentina
1950	8151	1228	8893	20	7	4118	6	46	17151
1955	8892	1393	9842	20	7	4424	6	52	18928
1960	9830	1624	10910	21	9	4798	6	55	20617
1965	10998	1884	11964	25	14	5135	6	60	22284
1970	12431	2157	13932	28	20	5606	7	66	23963
1975	14133	2402	16141	30	27	6051	7	69	26082
1980	15045	2672	18807	33	34	7206	7	69	28370
1985	13120	2957	22009	39	45	8390	7	65	30672

8 rows × 227 columns

In [ ]:

__Part 2. Exploring the data__

Now let's look at the data at hand.

2.1 Plotting population (15 pts)

Pick 6 countries of your choice and plot their population growth.

In [13]:

```
# Your code here
#Mexico, US, Guatemala
import matplotlib.pyplot as plt
plt.xlabel('Year')
plt.ylabel('Population')
#first Country:
plt.xticks(rotation= 45)
plt.plot(df.index,df.iloc[:,df.columns.get_loc('Mexico')].astype(int), label = 'Mexico')
plt.plot(df.index,df.iloc[:,df.columns.get_loc('Brazil')].astype(int), label = 'Brazil')
plt.plot(df.index,df.iloc[:,df.columns.get_loc('United States')].astype(int), label = 'Unit
plt.plot(df.index,df.iloc[:,df.columns.get_loc('Thailand')].astype(int), label = 'Thailand'
plt.plot(df.index,df.iloc[:,df.columns.get_loc('Japan')].astype(int), label = 'Japan')
plt.legend()
plt.show()
```

Out[13]:

```
Text(0.5, 0, 'Year')
```

Out[13]:

```
Text(0, 0.5, 'Population')
```

Out[13]:

```
(array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x12845f42070>]
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x12843f20fa0>]
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x12845f4b760>]
```

Out[13]:

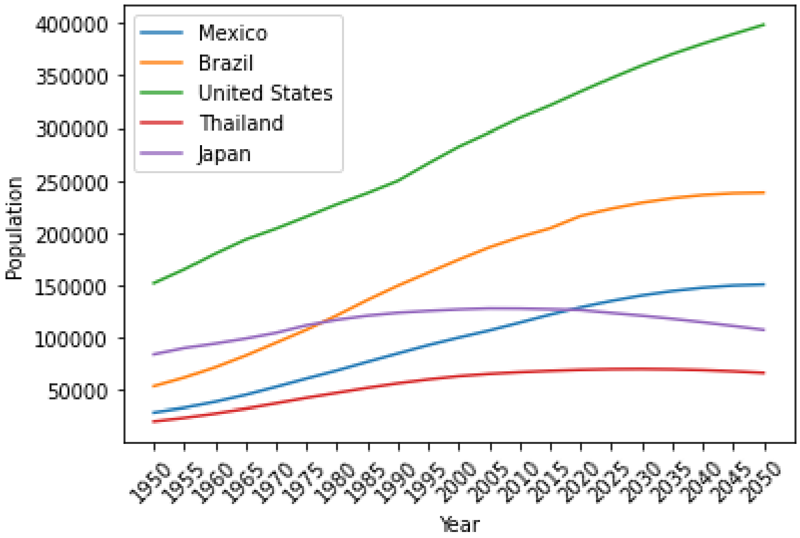
```
[<matplotlib.lines.Line2D at 0x12845f4ba90>]
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x12845f4be50>]
```

Out[13]:

```
<matplotlib.legend.Legend at 0x12843346a00>
```



2.2 Find 10 most populous countries (15 pts)

Find 10 most popoulous coutntries in 1960, 1980, 2000, 2020, and 2040. Plot and compare their population.

In [14]:

```
# Your code here
#plot for 1960
plt.title('Graph 1960')
plt.xlabel('Country')
plt.ylabel('Population')
row_1960 = df.loc['1960'].astype(int)
top = row_1960.nlargest(10)
plt.xticks(rotation= 45)
graph1 = plt.bar(top.keys(),top.iloc[0:,])
```

Out[14]:

```
Text(0.5, 1.0, 'Graph 1960')
```

Out[14]:

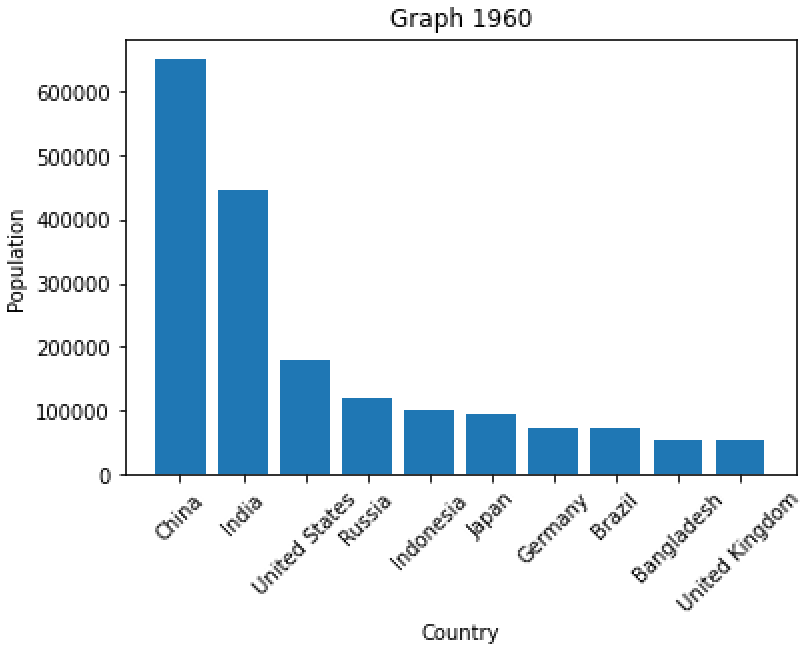
```
Text(0.5, 0, 'Country')
```

Out[14]:

```
Text(0, 0.5, 'Population')
```

Out[14]:

```
(array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



In [15]:

```
#plot for 1980
plt.title('Graph 1980')
plt.xlabel('Country')
plt.ylabel('Population')
row_1980 = df.loc['1980'].astype(int)
top = row_1980.nlargest(10)
plt.xticks(rotation= 45)
graph2 = plt.bar(top.keys(),top.iloc[0:,])
```

Out[15]:

```
Text(0.5, 1.0, 'Graph 1980')
```

Out[15]:

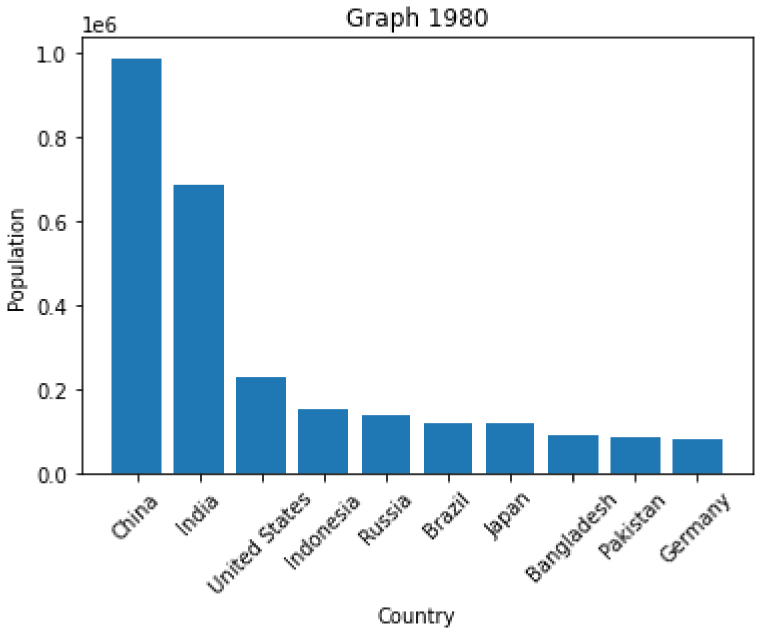
```
Text(0.5, 0, 'Country')
```

Out[15]:

```
Text(0, 0.5, 'Population')
```

Out[15]:

```
(array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



In [16]:

```
#plot for 2000
plt.title('Graph 2000')
plt.xlabel('Country')
plt.ylabel('Population')
row_2000 = df.loc['2000'].astype(int)
top = row_2000.nlargest(10)
plt.xticks(rotation= 45)
graph1 = plt.bar(top.keys(),top.iloc[0:,])
```

Out[16]:

```
Text(0.5, 1.0, 'Graph 2000')
```

Out[16]:

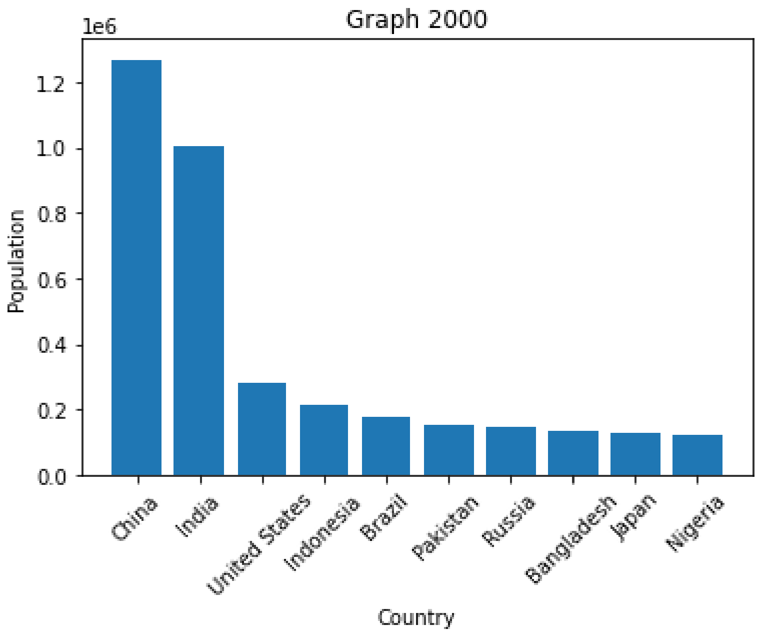
```
Text(0.5, 0, 'Country')
```

Out[16]:

```
Text(0, 0.5, 'Population')
```

Out[16]:

```
(array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



In [17]:

```
#2020
plt.title('Graph 2020')
plt.xlabel('Country')
plt.ylabel('Population')
row_2020 = df.loc['2020'].astype(int)
top = row_2020.nlargest(10)
plt.xticks(rotation= 45)
graph1 = plt.bar(top.keys(),top.iloc[0:,])
```

Out[17]:

```
Text(0.5, 1.0, 'Graph 2020')
```

Out[17]:

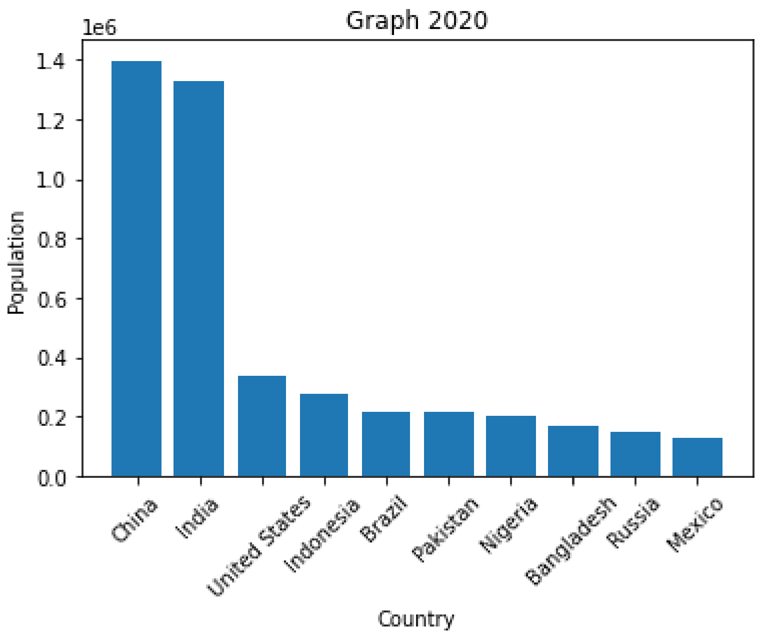
```
Text(0.5, 0, 'Country')
```

Out[17]:

```
Text(0, 0.5, 'Population')
```

Out[17]:

```
(array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



In [26]:

```
#2040
plt.title('Graph 2040')
plt.xlabel('Country')
plt.ylabel('Population')
row_2040= df.loc['2040'].astype(int)
top = row_2040.nlargest(10)
plt.xticks(rotation= 45)
graph1 = plt.bar(top.keys(),top.iloc[0:,])
```

Out[26]:

```
Text(0.5, 1.0, 'Graph 2040')
```

Out[26]:

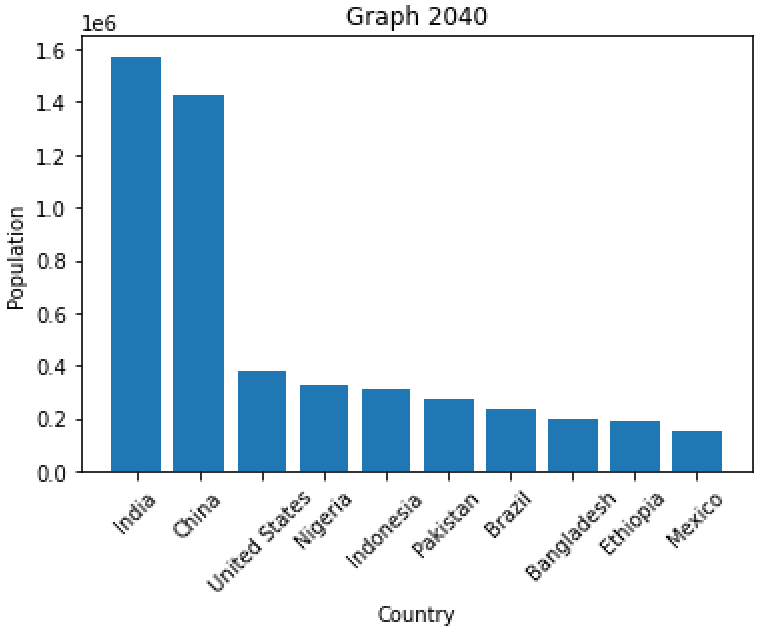
```
Text(0.5, 0, 'Country')
```

Out[26]:

```
Text(0, 0.5, 'Population')
```

Out[26]:

```
(array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
 <a list of 6 Text major ticklabel objects>)
```


Graph 2040
1e6
Population
India
China
United States
Nigeria
Indonesia
Pakistan
Brazil
Bangladesh
Ethiopia
Mexico
Country