Name: Nikita Bhrugumaharshi Emberi

STA 220 Assignment 4

Due **March 8, 2024** by **11:59pm**. Submit your work by uploading it to Gradescope through Canvas.

Instructions:

1. Provide your solutions in new cells following each exercise description. Create as many new cells as necessary. Use code cells for your Python scripts and Markdown cells for explanatory text or answers to non-coding questions. Answer all textual questions in complete sentences.
2. The use of assistive tools is permitted, but must be indicated. You will be graded on you proficiency in coding. Produce high quality code by adhering to proper programming principles.
3. Export the .jpynb as .pdf and submit it on Gradescope in time. To facilitate grading, indicate the area of the solution on the submission. Submissions without indication will be marked down. No late submissions accepted.
4. If test cases are given, your solution must be in the same format.
5. The total number of points is 10.

Exercise 1 Lets retrieve data from the CIA World Factbook and visualize parts of it.

(a) Using devtools, find a way to retrieve the names of all listed world entities. In order to navigate to their respective site, I assembled the path by processing the country names. To this end, **(i)** write a function `process_names` that processes the name as string according to the requests query parameter. *Run:*

```
process_names('Falkland Islands (Islas Malvinas)')
```

Importing necessary libraries

In [2]:
```
import re
import requests
import lxml.html as lx
import time
import pandas as pd
import numpy as np
from tqdm import tqdm
import pickle
import folium
from folium.plugins import MarkerCluster
import matplotlib.pyplot as plt
import nltk
from nltk.tokenize import word_tokenize
import geopandas as gpd
import plotly.express as px
import bokeh.io

from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.common.exceptions import TimeoutException
```

(i) write a function `process_names` that processes the name as string according to the requests query parameter. *Run:*

```
process_names('Falkland Islands (Islas Malvinas)')
```

In [3]:

```python
def process_names(name):
    # Remove characters other than letters, spaces, and hyphens
    cleaned_text = re.sub(r'[^a-zA-Z\s\-]+', '', name)

    # Replace double spaces with single spaces
    cleaned_text = re.sub(r'  ', ' ', cleaned_text)

    # Split the cleaned text into a list of words
    cleaned_text = cleaned_text.split(" ")

    # Remove the last element if it's an empty string
    if cleaned_text[-1] == '':
        cleaned_text = cleaned_text[:-1]

    # Join the words with hyphens and convert to lowercase
    output = '-'.join(word.lower() for word in cleaned_text)
    return output
```

Sample Given:

In [4]:

```python
print(process_names('French Southern and Antarctic Lands')) #french-southern-and-antarctic-lands
print(process_names('Bahamas, The')) #bahamas-the
```

```
french-southern-and-antarctic-lands
bahamas-the
```

Final Answer:

In [5]:

```python
process_names('Falkland Islands (Islas Malvinas)')
```

Out[5]:

```
'falkland-islands-islas-malvinas'
```

(ii) Obtain all world entity names. *How many have you found? Hint: I could not retrieve data for all 266 entities that the CIA WFB claims to have.*

In [9]:

```python
url = "https://www.cia.gov/the-world-factbook/countries/"

# Sending a GET request to the specified URL
result = requests.get(url)

# Raising an exception if the request isn't successful
result.raise_for_status()
```

In [10]:

```python
# Initializing a Chrome webdriver instance
driver = webdriver.Chrome()

# Setting a page load timeout of 20 seconds
driver.set_page_load_timeout(20) # twenty seconds should be enough
```

```
try:
    # Attempt to load the URL in the browser
    driver.get(url)
except TimeoutException:
    # If the page load times out, stop loading the page
    driver.execute_script("window.stop();")
```

Final Answer:

In [11]:

```
# Initializing an empty list to store page URLs
world_entities = []

# loop until we reach the last page of countries page
while True:
    try:
        time.sleep(0.2)

        # Parsing the current page's HTML source
        html = lx.fromstring(driver.page_source)

        # Extracting the <div> element containing world entities
        row_element = html.xpath('//*[@id="index-content-section"]/div/div[2]/div[1]')[0]

        # Extracting names inside the row element
        div_elements = row_element.xpath('./div/div/h2/a')

        # Extending the list of world_entities with the extracted country names
        for name in div_elements:
            world_entities.append(name.text)

        # Finding the element to click on it to go to the next page
        next_page = driver.find_element("xpath", "//span[@class='pagination__arrow-right']")

        next_page.click()

    except:
        break

print(f"Number of world entity names found: {len(world_entities)}")
```

```
Number of world entity names found: 262
```

In [12]:

```
# Closing Selenium window
driver.quit()
```

(iii) Write a function `get_info` takes a country name as string as input and return all the information as json that is displayed on its respective site. Use the retrieved data set for the next exercises. *Hint: If you rate-limit your requests (you should!) this may take up to 10 minutes.*

In [13]:

```
import requests
import lxml.html as lx

def get_info(country_name):
    country_dict = {}
    try:
        processed_name = process_names(country_name)  # Process the country name

        # Construct the URL
        url = "https://www.cia.gov/the-world-factbook/countries/" + processed_name
        result = requests.get(url)

        # Parse the HTML content
        html = lx.fromstring(result.text)

        # Find the main content section
        main_content = html.xpath('//*[@id="main-content"]/section[3]/div/div/div[2]')

        # If main content not found, handle the exception and try another URL (default URL on the site for certain islands)
        if len(main_content) == 0:
            url = "https://www.cia.gov/the-world-factbook/countries/united-states-pacific-island-wildlife-refuges/"
            result = requests.get(url)
            html = lx.fromstring(result.text)
            main_content = html.xpath('//*[@id="main-content"]/section[3]/div/div/div[2]')

        # Get the first element of the main content list
        main_content = main_content[0]

        # Find all h3 elements within main content
        titles = main_content.xpath('//h3')

        for title in titles:
            try:
                 # Find sibling p elements
                values = title.xpath('./following-sibling::p')

                # Initialize an empty string to store the combined text
                combined_text = ''
                for value in values:
                    for text in value.xpath('.//text()'):
                        # Concatenate the stripped text to the combined_text string along with a newline character
                        combined_text += ''.join(text).strip() + '\n'

                # Store the combined text in the dictionary
                country_dict[title.text_content()] = combined_text
            except Exception as e:
                print(f"An error occurred while processing {title.text_content()}: {e}")

    except Exception as e:
        print(f"An error occurred while fetching data for {country_name}: {e}")  # Print error message for exception

    # Return the populated dictionary
    return country_dict
```

In [14]:
```
country_data = {}

for country in tqdm(world_entities):
    country_data[country] = get_info(country)
```

```
100%|█████████████████████████████████████████| 262/262 [01:24<00:00,  3.10it/s]
```

Final Answer

In [16]:

```python
# showing only first few key, value pairs of the obtained data so that it doesn't clutter the output cell after converting to PDF
my_dict = country_data['United States']
keys_to_display = ['Background', 'Location', 'Geographic coordinates', 'Map references', 'Climate']

for key in keys_to_display:
    if key in my_dict:
        print(f"{key}: {my_dict[key]}")
    else:
        print(f"Key '{key}' not found in the dictionary.")
```

```
Background: Britain's American colonies broke with the mother country in 1776 and were recognized as the new nation of the United States of America following the Treaty of Paris in
1783. During the 19th and 20th centuries, 37 new states were added to the original 13 as the nation expanded across the North American continent and acquired a number of overseas po
ssessions. The two most traumatic experiences in the nation's history were the Civil War (1861-65), in which a northern Union of states defeated a secessionist Confederacy of 11 sou
thern slave states, and the Great Depression of the 1930s, an economic downturn during which about a quarter of the labor force lost its jobs. Buoyed by victories in World Wars I an
d II and the end of the Cold War in 1991, the US remains the world's most powerful nation state. Since the end of World War II, the economy has achieved relatively steady growth, lo
w unemployment and inflation, and rapid advances in technology.

Location: North America, bordering both the North Atlantic Ocean and the North Pacific Ocean, between Canada and Mexico

Geographic coordinates: 38 00 N, 97 00 W

Map references: North America

Climate: mostly temperate, but tropical in Hawaii and Florida, arctic in Alaska, semiarid in the great plains west of the Mississippi River, and arid in the Great Basin of the south
west; low winter temperatures in the northwest are ameliorated occasionally in January and February by warm chinook winds from the eastern slopes of the Rocky Mountains
note:
many consider Denali, the highest peak in the US, to be the worldâ€  s coldest mountain because of its combination of high elevation and its subarctic location at 63 degrees north l
atitude; permanent snow and ice cover over 75 percent of the mountain, and enormous glaciers, up to 45 miles long and 3,700 feet thick, spider out from its base in every direction;
it is home to some of the worldâ€  s coldest and most violent weather, where winds of over 150 miles per hour and temperatures of -93Ë  F have been recorded. Â
```

(b) Lets learn about the newest updated data points in the CIA world factbook - the merchant marine! **(i)** Write a function `ports` that returns a list of all major seaports of a given country. *Run:*

```
ports('United States')
```

In [31]:

```python
# Following list of items could appear after mentioning info about major seaports on the site
to_skip = ['river port(s):',
           'LNG terminal(s) (export):',
           'cruise port(s):',
           'container port(s) (TEUs):',
           'LNG terminal(s) (import):',
           'oil terminal(s):',
           'dry bulk cargo port(s):',
           'lake port(s):',
           'river and lake port(s):',
           'river or lake port(s):',
           'bulk cargo port(s):',
           'major port(s):',
           'cruise/ferry port(s):',
           'Saint Helena:',
           'Ascension Island:',
           'Tristan da Cunha:',
           'note:',
           'cargo ports:',
           'cruise departure ports (passengers):']


def ports(country):
    selected_lines = []
    start_flag = False
    seaPorts = []

    data = country_data[country]
    if "Ports and terminals" in data:
        text = data['Ports and terminals']
        text = re.sub(r'\n+', '\n', text)
        lines = text.split("\n")

        # Check if 'Ports and terminals' information is available for the country
        for line in lines:
            if start_flag:
                # In order to extract information about major sepaorts only
                # Check if any skip phrase is present in the line
                if any(skip_phrase in line for skip_phrase in to_skip):
                    break
                selected_lines.append(line)
            elif 'major seaport(s):' in line:
                start_flag = True
                selected_lines.append(line)

        for item in selected_lines:
            if item.endswith(":"):
                continue
            elif "," in item or "/" in item:
                seaPorts.extend([x.strip() for x in re.split(r'[,/]', item)])
            else:
                seaPorts.append(item.strip())
    seaPorts = [item for item in seaPorts if item]
    return seaPorts
```

Final Answer:

In [18]:

```
ports('United States')
```

Out[18]:

```
['Charleston',
 'Hampton Roads',
 'New York',
 'New Jersey',
 'Savannah',
 'Long Beach',
 'Los Angeles',
 'Oakland',
 'Seattle',
 'Tacoma',
 'Houston']
```

(ii) Lets put a marker on a world map corresponding to the location of all major seaports that you retrieved. Use the Nominatim API to get latitute-longitude pairs. Make structured queries and pass the `city` and `country` keys. Use the first value that is returned.

Print the world map. Name three markers that are apparently misplaced.

In [19]:

```python
# Function that makes structured query and returns latitude and longitude pairs based on country and city of ports
def lat_lon(country, city):
    pairs = [0,0]
    url = 'https://nominatim.openstreetmap.org/search'

    # Example parameters
    params = {
        'country': country,
        'city': city,
        'format' : 'json'
    }

    # Making the GET request with parameters
    response = requests.get(url, params=params)

    # Checking if the request was successful (status code 200)
    if response.status_code == 200:
        # Extracting the response data
        data = response.json()
        # Process the data as needed for longitude and latitude
        if len(data) != 0:
            pairs[0] = data[0]['lat']
            pairs[1] = data[0]['lon']
    else:
        print("Request failed with status code:", response.status_code)

    return pairs
```

In [20]:

```python
# Preparing data for map plotting:

map_data = {}
for country in tqdm(world_entities):
    port_loc_list = []
    seaPorts = ports(country)
    if len(seaPorts) > 0:
        for port in seaPorts:
            loc_pair = lat_lon(country, port)
            if loc_pair != [0,0]:
                port_loc_list.append([port, loc_pair[0], loc_pair[1]])
    map_data[country] = port_loc_list
```

```
100%|████████████████████████████████████████| 262/262 [05:13<00:00,  1.20s/it]
```

In [21]:

```
# Sample Data prepared
map_data['United States']
```

Out[21]:

```
[['Charleston', '32.7884363', '-79.9399309'],
 ['Hampton Roads', '36.9507552', '-76.41141865690076'],
 ['New York', '40.7127281', '-74.0060152'],
 ['New Jersey', '40.8326131', '-74.4323251'],
 ['Savannah', '32.0790074', '-81.0921335'],
 ['Long Beach', '33.7690164', '-118.191604'],
 ['Los Angeles', '34.0536909', '-118.242766'],
 ['Oakland', '37.8044557', '-122.271356'],
 ['Seattle', '47.6038321', '-122.330062'],
 ['Tacoma', '47.2455013', '-122.438329'],
 ['Houston', '29.7589382', '-95.3676974']]
```

In [22]:

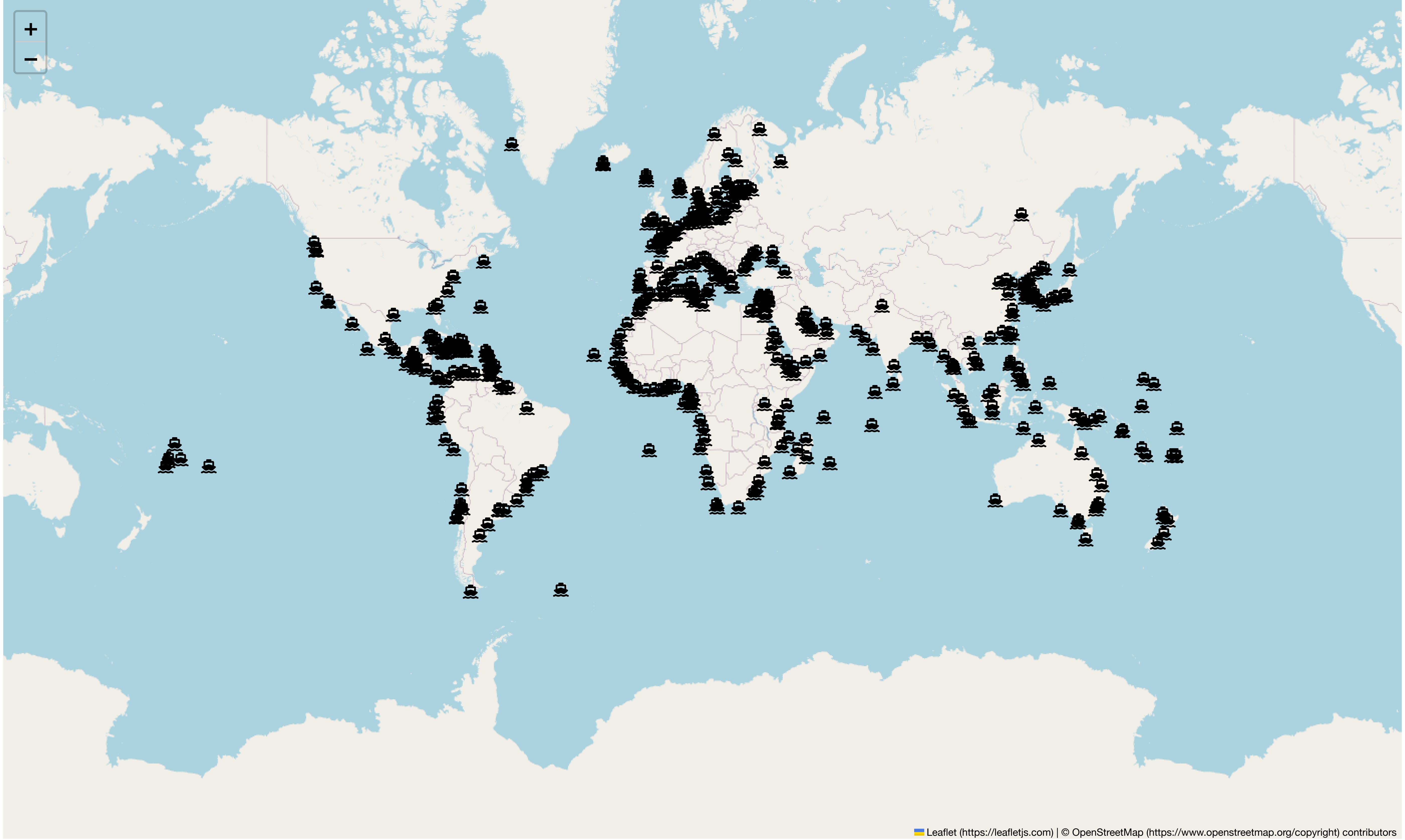
```
plot_data = []

for country, ports in map_data.items():
    if len(ports) > 0:
        for port in ports:
            plot_dict = {}
            plot_dict['city'] = port[0]
            plot_dict['lat'] = port[1]
            plot_dict['lon'] = port[2]
            plot_data.append(plot_dict)
```

Final Answer:

In [98]:

```
# Create a map centered on a location
world_map = folium.Map(location=[0, 0], zoom_start=2)

# Add markers for each seaport to the map
for seaport in plot_data:
    folium.Marker(
        location=[seaport["lat"], seaport["lon"]],
        popup=seaport["city"],
        icon=folium.Icon(icon="ship", prefix="fa", icon_size=(12, 12), icon_color="olive", shadow_size = (0,0), icon_anchor=(12, 12))  # Reduce marker size
    ).add_to(world_map)


# Display the map inline
%matplotlib inline
world_map
```

Out[98]:


+
−
Leaflet (https://leafletjs.com) | © OpenStreetMap (https://www.openstreetmap.org/copyright) contributors

In [24]:

```
print('''
The three markers that are apparently misplaced are:
1) Kandla, India
2) Vostochnyy, Russia
3) Bandar-e Emam Khomeyni, Iran
```

```
------------------------------------------------------
4) San Vicente, Chile (Also Misplaced)
''')
```

```
The three markers that are apparently misplaced are:
1) Kandla, India
2) Vostochnyy, Russia
3) Bandar-e Emam Khomeyni, Iran
------------------------------------------------------
4) San Vicente, Chile (Also Misplaced)
```

(iii) Amongst all countries with a major seaport, return the four that have the largest fleet of *bulk carriers*. Amongst all countries with no coastline, return the four that have the largest merchant marine fleet overall.

In [32]:

```python
# Function to check whether a country has major Sea Ports
def has_major_seaport(country):
    seaPorts = ports(country)
    if len(seaPorts) > 0:
        return True
    return False
```

In [33]:

```python
# Function to check if a country has coastline
def has_no_coastline(country):
    coastline_info = country_data[country]
    if 'Coastline' in coastline_info:
        coastline_info = coastline_info['Coastline']
        # Regular expression to match the number before "km"
        pattern = r'(\d{1,3}(?:,\d{3})*)(?=\s*km)'
        matches = re.findall(pattern, coastline_info)
        if matches:
            # Extract the first match (assuming there's only one match)
            number = matches[0]
            # Remove commas from the number and convert it to an integer
            number = int(number.replace(',', ''))

            if number > 0:
                return False
    return True
```

In [34]:

```python
def bulk_carrier(country):
    text = country_data[country]

    # Check if 'Merchant marine' information is available for the country
    if 'Merchant marine' in text:
        text = text['Merchant marine']
        text = re.sub(r'\n+', '\n', text)
        lines = text.split("\n")

        # Check if 'by type:' is present in the lines
        if "by type:" in lines:
            index = lines.index('by type:')
            cargos = lines[index + 1].split(", ")

            # Iterate through the list of cargos
            for item in cargos:
                if 'bulk carrier' in item:
                    # Extract the number of bulk carriers
                    number = item.split()[2]
                    number = int(number.replace(',', ''))
                    return number

    return 0
```

In [35]:

```
def merchant_marine(country):
    text = country_data[country]

    # Check if 'Merchant marine' information is available for the country
    if 'Merchant marine' in text:
        text = text['Merchant marine']
        text = re.sub(r'\n+', '\n', text)
        lines = text.split("\n")

        # Check if 'total:' is present in the lines
        if "total:" in lines:
            index = lines.index('total:')
            cargos = lines[index + 1].split(" ")
            return int(cargos[0].replace(',', ''))

    return 0
```

In [36]:

```
countries_with_seaport = {}
countries_with_no_coastline = {}

for country in tqdm(world_entities):
    if has_major_seaport(country):
        countries_with_seaport[country] = bulk_carrier(country)
    if has_no_coastline(country):
        countries_with_no_coastline[country] = merchant_marine(country)
```

```
100%|██████████████████████████████████████| 262/262 [00:00<00:00, 34997.06it/s]
```

Final Answer: Amongst all countries with a major seaport, return the four that have the largest fleet of bulk carriers

In [37]:

```
sorted_bulk_carrier = sorted(countries_with_seaport.items(), key=lambda x: x[1], reverse=True)
print("Top 4 countries with a major seaport that have the largest fleet of bulk carriers: ")
print("==================================================================================== ")
for key, value in sorted_bulk_carrier[:4]:
    print(f"{key} : ({value})")
```

```
Top 4 countries with a major seaport that have the largest fleet of bulk carriers:
====================================================================================
Panama : (2732)
Marshall Islands : (1939)
Liberia : (1895)
China : (1831)
```

Final Answer: Amongst all countries with no coastline, return the four that have the largest merchant marine fleet overall.

In [38]:
```
sorted_merchant_marine = sorted(countries_with_no_coastline.items(), key=lambda x: x[1], reverse=True)
print("Top 4 countries with no coastline that have the largest merchant marine fleet overall: ")
print("===================================================================================== ")
for key, value in sorted_merchant_marine[:4]:
    print(f"{key} : ({value})")
```

```
Top 4 countries with no coastline that have the largest merchant marine fleet overall:
=====================================================================================
Mongolia : (318)
Azerbaijan : (312)
Luxembourg : (147)
Kazakhstan : (122)
```

(c) Now, lets classify whether a country is or has been controlled by the United Kingdom by analyzing the provided background information text. **(i)** Implement a (very simple!) classification method that performs this task. My function `was_british` correctly identifies the countries of Pakistan and Russia, but incorrectly classifies Spain and the United States.

How many world entities do you find to be current or former parts of the British Empire?

In [39]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/nikitabhrugumaharshiemberi/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[39]:
```
True
```

In [40]:
```
def was_british(country):
    background = country_data[country]['Background']

    # keywords that could indicate the presence of British Control
    keywords = ["british empire", "colonial rule", "british colony", "british protectorate", "formerly british",
                "administered by britain", "british control", "british mandate", "uk rule", "uk colony",
               "british", "colonial", "colony", "colonies" ,"colonized by", "ruled by",
                "controlled by", 'occupied by', "ruled by britain", "ruled by uk"]
    background_tokens = word_tokenize(background)
    lowercase_bg_tokens = [token.lower() for token in background_tokens]
    is_controlled_by_uk = any(keyword in lowercase_bg_tokens for keyword in keywords)
    return is_controlled_by_uk
```

Final Answer: Countries correctly classified in provided sample and my classifier

In [41]:
```
print(was_british('Pakistan')) # True
print(was_british('Russia')) # False
```

```
True
False
```

Final Answer: Countries misclassified in provided sample but correctly classified by my classifier

In [42]:
```
print(was_british('Spain')) # True
print(was_british('United States')) # False
```

```
False
True
```

In [43]:

```
colonies = []
for country in world_entities:
    if was_british(country):
        colonies.append(country)
```

In [44]:

```
# Since "world" is not a country
colonies.remove('World')
```

Final Answer:

In [45]:

```
print(f"Number of world entities that were found to to be current or former parts of the British Empire: {len(colonies)}")
```

```
Number of world entities that were found to to be current or former parts of the British Empire: 136
```

(ii) Retrieve the ISO codes from here and use them to color all countries on a world map that you have determined to be former parts of the British Empire. The map should look something like this.

In [47]:

```
colonies_with_iso = []

# URL to fetch country data codes
url = "https://www.cia.gov/the-world-factbook/references/country-data-codes/"
result = requests.get(url)
html = lx.fromstring(result.text)

# Find the main table containing country data codes
main_table = html.xpath('//table[@class = "content-table table-auto"]')[0]
table_body = main_table.xpath('./tbody/tr')

# Iterate through each row in the table
for tr in table_body:
    data = tr.xpath('.//text()')  # Extract text from each cell in the row

    # Check if the country name is present in the list of colonies
    if data[0] in colonies:
        colony_info = {}
        colony_info['country'] = data[0]
        colony_info['iso'] = data[1]
        colonies_with_iso.append(colony_info)
```

In [48]:

```
# manually adding iso value for which code cannot be fetched
colonies_with_iso.extend([{
    'country' : 'South Georgia and South Sandwich Islands',
    'iso' : 'SGS'
    },
    {'country' : 'United States Pacific Island Wildlife Refuges',
    'iso' : 'UM'}])
```

In [49]:

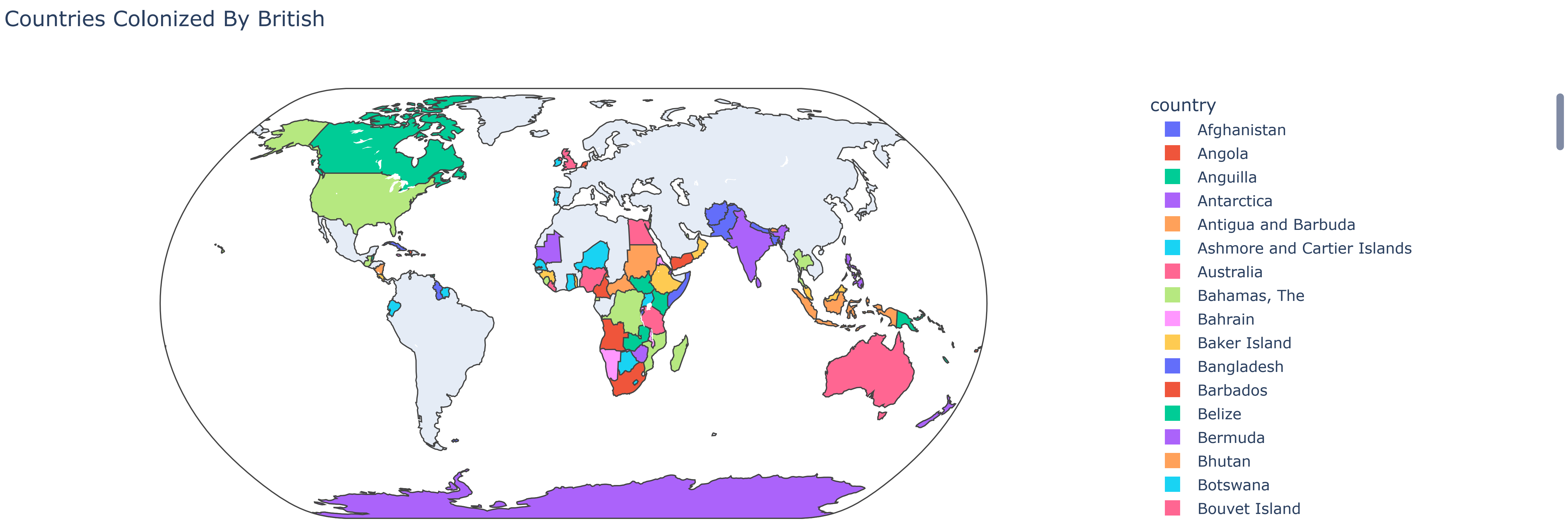
```
# Convert the list of dictionaries to a DataFrame
df = pd.DataFrame(colonies_with_iso)

# Rename the columns as required
df.columns = ['country', 'iso']
df.head(4)
```

Out[49]:

	country	iso
0	Afghanistan	AFG
1	Angola	AGO
2	Anguilla	AIA
3	Antarctica	ATA

In [51]:

```
fig = px.choropleth(df, locations='iso', color = 'country', hover_name='country', locationmode='ISO-3',
                    projection='natural earth', title='Countries Colonized By British', color_continuous_scale='lightpink')  # Specify text to display on hover

fig.show(renderer='iframe')
```



(d) Lets build our own population pyramide (with only three steps) according to the obtained data (`0-14`, `15-64`, `65+`). Given the current health expenditure as threshold, we want to obtain create a population pyramide for all aggregated population values. **(i)** Assemble a data frame that given a threshold shows the aggregated population values of all data points which current health expenditure does not exceed the threshold, separated by gender. The first four rows of data frame `df` are given below.

How many distinct thresholds do you find?

In [52]:

```
def get_population_info(str):
    pattern = r'male ([0-9,]+)|female ([0-9,]+)'

    # Find matches using the pattern
    matches = re.findall(pattern, str)

    # Initialize dictionary to store male and female populations
    populations = {'male': 0, 'female': 0}

    # Extract male and female populations from matches
    for match in matches:
        for i in range(len(match)):
            if match[i]:
                populations['male' if i == 0 else 'female'] = int(match[i].replace(',', ''))

    return populations
```

In [53]:

```
# Initialize lists to store distinct health expenditure values and population pyramid data
distinct_health_exp = []
pyramid_data = []

# Define age groups
groups = ['0-14 years', '15-64 years', '65 years and over']

# Iterate through each country
for country in tqdm(world_entities):
    data = country_data[country]  # Get data for the current country

    # Check if 'Current health expenditure' and 'Age structure' information is available
    if 'Current health expenditure' in data and 'Age structure' in data:
        health_exp = data['Current health expenditure'].split()  # Split health expenditure data
        text = country_data[country]['Age structure']  # Get age structure data
        lines = text.split("\n")  # Split age structure data into lines
        lines.remove('')  # Remove empty lines

        # Convert health expenditure to float if it's not 'NA'
        if health_exp[0] != 'NA':
            health_exp = float(health_exp[0].rstrip('%'))  # Remove '%' and convert to float

            # Iterate through age groups
            for i in range(3):
                dict = {}  # Initialize dictionary to store data
                dict['country'] = country  # Add country name to dictionary
                dict['Groups'] = groups[i]  # Add age group to dictionary
                population_info = get_population_info(lines[2*i + 1])  # Extract population info
                dict['Male'] = population_info.get('male')  # Add male population to dictionary
                dict['Female'] = population_info.get('female')  # Add female population to dictionary
                dict['threshold'] = health_exp  # Add health expenditure threshold to dictionary
                pyramid_data.append(dict)  # Append dictionary to pyramid_data list

            # Add distinct health expenditure value to list if not already present
            if health_exp not in distinct_health_exp:
                distinct_health_exp.append(health_exp)
```

```
100%|██████████████████████████████████████| 262/262 [00:00<00:00, 55838.80it/s]
```

In [54]:

```
distinct_health_exp.sort()
```

In [55]:

```
final_pyramid_data = []

# Iterate through distinct health expenditure values
for threshold in distinct_health_exp:
    group_1_Male = 0
    group_2_Male = 0
    group_3_Male = 0

    group_1_Female = 0
    group_2_Female = 0
    group_3_Female = 0

    # Iterate through pyramid data to calculate population for each age group that qualifies the given threshold value
    for data in pyramid_data:
        if data['threshold'] >= threshold:
            if data['Groups'] == '0-14 years':
                group_1_Male += data['Male']
                group_1_Female += data['Female']
            elif data['Groups'] == '15-64 years':
                group_2_Male += data['Male']
                group_2_Female += data['Female']
            elif data['Groups'] == '65 years and over':
                group_3_Male += data['Male']
                group_3_Female += data['Female']

    # Create dictionaries for each age group with calculated population and threshold
    dict = [{'Groups': '0-14 years', 'Male': group_1_Male, 'Female': group_1_Female, 'Threshold': threshold},
            {'Groups': '15-64 years', 'Male': group_2_Male, 'Female': group_2_Female, 'Threshold': threshold},
            {'Groups': '65 years and over', 'Male': group_3_Male, 'Female': group_3_Female, 'Threshold': threshold}]

    # Extend final_pyramid_data with the dictionaries
    final_pyramid_data.extend(dict)
```

Final Answer

In [56]:

```
pyramid_df = pd.DataFrame(final_pyramid_data, columns=['Groups', 'Male', 'Female', 'Threshold'])
pyramid_df.head(6)
```

Out[56]:

	Groups	Male	Female	Threshold
0	0-14 years	999784527	940797401	1.7
1	15-64 years	2595737039	2532780848	1.7
2	65 years and over	356032162	441482834	1.7
3	0-14 years	999783005	940795953	2.0
4	15-64 years	2595728401	2532772290	2.0
5	65 years and over	356027048	441476517	2.0

In [57]:

```
print(f"Number of distinct thresholds found: {len(distinct_health_exp)}")
```

```
Number of distinct thresholds found: 92
```

(ii) Using `bokeh.io`, create a client-based interactive opulation pyramid that displays the data from (i) according to a set threshold (or the closest threshold that exists). Make sure that the pyramid is well crafted, similar to this, but with a slider and only three population groups.

Either provide a link to a site that hosts the interactive graphic, or provide a non-interactive for threshold value `10`.

In [72]:

```
bokeh.io.output_notebook()
```

BokehJS 3.3.4 successfully loaded.

In [86]:

```
from bokeh.plotting import figure, show, curdoc
from bokeh.layouts import Column
from bokeh.models import Slider, CustomJS, ColumnDataSource, CDSView, GroupFilter, HoverTool
```

In [87]:

```
plot_df = pyramid_df.copy()
# multiplying by "-1" so that values for Male are plotted in the left direction to match the sample image provided
plot_df['Male'] = -1 * plot_df['Male']
plot_df.head()
```

Out[87]:

	Groups	Male	Female	Threshold
0	0-14 years	-999784527	940797401	1.7
1	15-64 years	-2595737039	2532780848	1.7
2	65 years and over	-356032162	441482834	1.7
3	0-14 years	-999783005	940795953	2.0
4	15-64 years	-2595728401	2532772290	2.0

In [88]:

```
# Set up the slider.
start = plot_df["Threshold"].min()
end = plot_df["Threshold"].max()
slider = Slider(start = start, end = end, value = start)
```

In [76]:

```
type(slider)
```

Out[76]:

```
bokeh.models.widgets.sliders.Slider
```

In [77]:

```
groups = plot_df["Groups"].unique()
groups
```

Out[77]:

```
array(['0-14 years', '15-64 years', '65 years and over'], dtype=object)
```

In [89]:

```
# Set up figure.
p = figure(title = "Population Pyramid",
           width = 800, height = 400,
           y_range = groups)
p.xaxis.axis_label = "Population Values"
p.yaxis.axis_label = "Age Groups"
```

In [90]:

```
# Set up data sources.
plot_df["Threshold"] = plot_df["Threshold"].apply(lambda x: str(x))

source = ColumnDataSource(data = plot_df)

view = CDSView(filter = GroupFilter(
    column_name = "Threshold", group = str(start)))
```

In [91]:

```
# Add horizontal bars for male and female populations
p.hbar(y='Groups', right='Male', height=0.4, color='purple', source=source, view = view, legend_label='Male')
p.hbar(y='Groups', right='Female', height=0.4, color='pink', source=source, view = view, legend_label='Female')

# Add hover tool
hover = HoverTool()
hover.tooltips = [("Age Group", "@Groups"), ("Male Population", "@Male"),
                  ("Female Population", "@Female")]
p.add_tools(hover)


# Customize the plot
p.title.text_font_size = "16pt"
p.legend.orientation = "horizontal"
p.legend.location = "top_right"
```

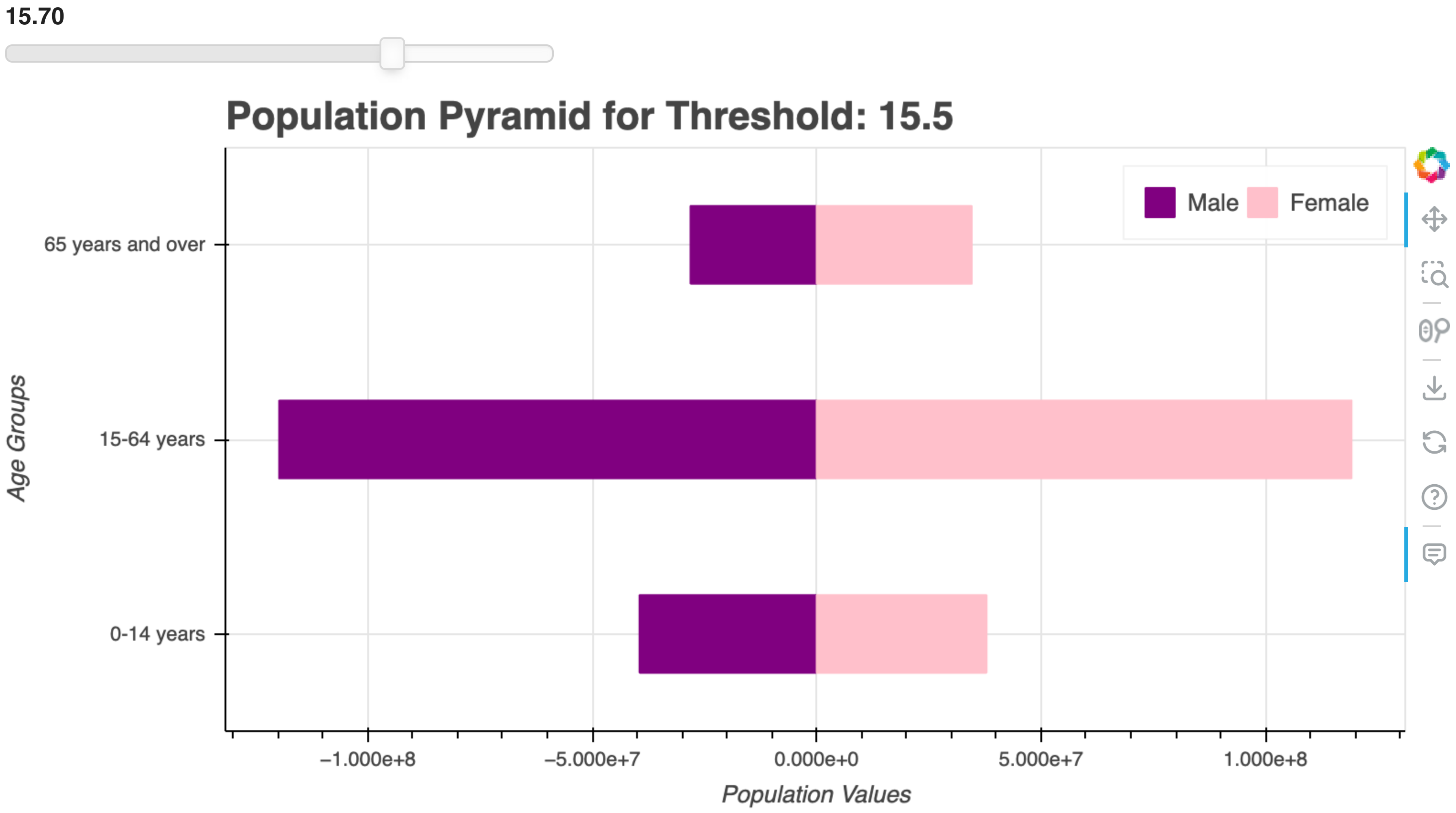
In [92]:

```
callback = CustomJS(
    args =
    {"source": source,
     "view": view,
     "figure": p,
     "thresholds" : plot_df['Threshold'].unique()
    },
    code = """
    let slider_start = cb_obj.value;
    let best_d = Math.abs(thresholds[0] - cb_obj.value);

    for (let threshold of thresholds) {
        let d = Math.abs(threshold - cb_obj.value);

        if (d < best_d) {
            slider_start = threshold;
```

```
                best_d = d;
            }
        }

        let slider_value = slider_start.toString();
        view.filters[0].group = slider_value;
        figure.title.text = "Population Pyramid for Threshold: " + slider_value;
        source.change.emit();
        """
)

slider.js_on_change("value", callback)
```

Final Answer (with html file hosted)

In [93]:
```
# Display the plot
layout = Column(slider, p)
show(layout)
```



In [94]:
```
# Optional: save the plot to a standalone HTML file.
bokeh.io.output_file("MY_PLOT.html")
```

In [95]:
```
from IPython.display import FileLink

# Provide the filename you want to attach
file_name = 'MY_PLOT.html'
```

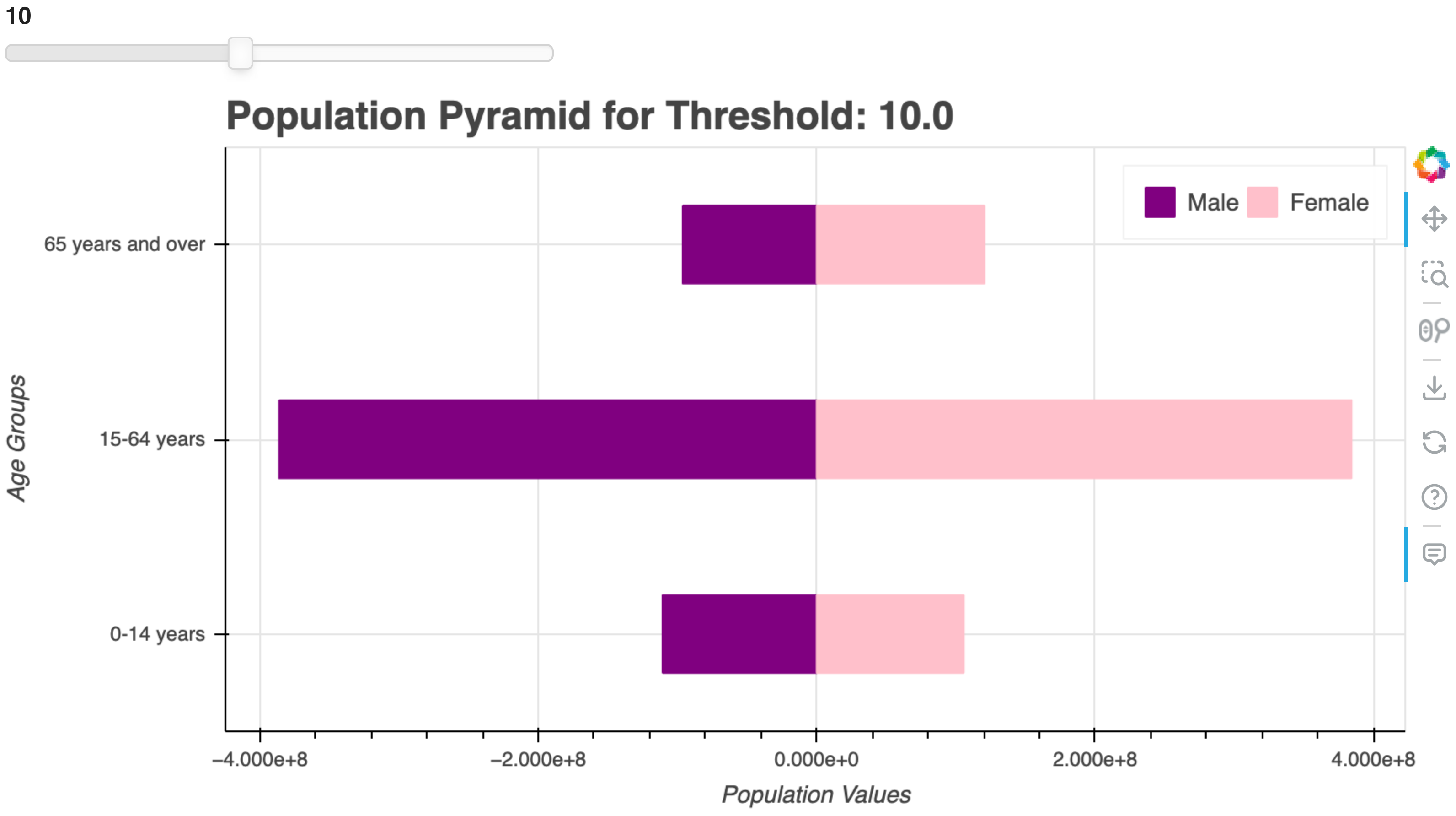
```
# Create a FileLink object with the filename
file_link = FileLink(file_name)

# Display the link
display(file_link)
```

MY_PLOT.html

Final Answer: (static plot with threshold 10)

In [96]:

```
slider = Slider(start = 1, end = 22, value = 1)

fig = figure(title = "Population Pyramid for Threshold: 10",
           width = 800, height = 400,
           y_range = groups)
fig.xaxis.axis_label = "Population Values"
fig.yaxis.axis_label = "Age Groups"

source = ColumnDataSource(data = plot_df)

view = CDSView(filter = GroupFilter(
    column_name = "Threshold", group = str(start)))


fig.hbar(y='Groups', right='Male', height=0.4, color='purple', source=source, view = view, legend_label='Male')
fig.hbar(y='Groups', right='Female', height=0.4, color='pink', source=source, view = view, legend_label='Female')

# Add hover tool
hover = HoverTool()
hover.tooltips = [("Age Group", "@Groups"), ("Male Population", "@Male"),
                  ("Female Population", "@Female")]
fig.add_tools(hover)


# Customize the plot
fig.title.text_font_size = "16pt"
fig.legend.orientation = "horizontal"
fig.legend.location = "top_right"

callback = CustomJS(
    args =
    {"source": source,
     "view": view,
     "figure": fig,
     "thresholds" : plot_df['Threshold'].unique()
    },
    code = """
    let slider_start = cb_obj.value;
    let best_d = Math.abs(thresholds[0] - cb_obj.value);

    for (let threshold of thresholds) {
        let d = Math.abs(threshold - cb_obj.value);

        if (d < best_d) {
            slider_start = threshold;
            best_d = d;
        }
    }

    let slider_value = slider_start.toString();
    view.filters[0].group = slider_value;
    figure.title.text = "Population Pyramid for Threshold: " + slider_value;
    source.change.emit();
    """
)

slider.js_on_change("value", callback)

# Display the plot
layout = Column(slider, fig)
show(layout)
```



Acknowledgment

I received assistance from `ChatGPT` while working on certain questions in this notebook. I want to clarify that I independently completed the majority of the tasks, seeking help only in instances where I encountered challenges or felt lost. The collaboration with ChatGPT was instrumental in providing guidance and insights during those moments. ChatGpt: https://chat.openai.com/

--- Nikita Bhrugumaharshi Emberi

References:

1. https://www.cia.gov/the-world-factbook/

In [ ]: