# 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)')
```

In [ ]:
```
import re

def process_names(input_string):
    # Convert the string to lowercase
    processed_string = input_string.lower()

    # Use regular expression to remove parentheses and replace spaces with '-'
    processed_string = re.sub(r'[\(\)]', '', processed_string)
    processed_string = re.sub(r'\s', '-', processed_string)

    return processed_string
```

In [ ]:
```
print(process_names('French Southern and Antarctic Lands'))
print(process_names('Bahamas, The'))
print(process_names('Falkland Islands (Islas Malvinas)'))
```

```
french-southern-and-antarctic-lands
bahamas,-the
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 [ ]:
```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import re
import json
import requests
from bs4 import BeautifulSoup
import time
```

In [ ]:
```python
# Set up the Selenium WebDriver with Firefox
driver = webdriver.Firefox()  # Ensure geckodriver is installed and in your PATH

# Open the website
driver.get("https://www.cia.gov/the-world-factbook/countries/")
name_list = []
for _ in range(23):
    next_arrow = WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.CSS
    next_arrow.click()
    elements = driver.find_elements(By.XPATH, '//a/.[@class="inline-link"]')

    for element in elements:
        if element not in name_list:
            name_list.append(process_names(element.text))
```

In [ ]:
```python
name_list = set(name_list)
name_list = list(name_list)
print(f"There are {len(name_list)} countries extracted")
```

```
There are 262 countries extracted
```

**(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 [ ]:
```
def get_info(country_name):
    base = "https://www.cia.gov/the-world-factbook/countries/"
    url = base + country_name
    bs = BeautifulSoup(requests.get(url).text)
    texts = bs.find_all('h3', {'class':"mt30"})
    info = {}
    for text in texts:
        title = text.text
        stuff = text.find_next_siblings()

        paragraphs = [tag.get_text(strip=True) for tag in stuff]
        single_paragraph = ' '.join(paragraphs)
        info[title] = single_paragraph

    # Join the paragraphs into a single paragraph
    return info

us = get_info("united-states")
```

In [ ]:

Out[ ]:
```
'major seaport(s):Atlantic Ocean:Charleston, Hampton Roads, New York/New Jersey, Sa
vannahPacific Ocean:Long Beach, Los Angeles, Oakland, Seattle/TacomaGulf of Mexico:
Houstonoil terminal(s):LOOP terminal, Haymark terminalcontainer port(s) (TEUs):Char
leston (2,751,442), Hampton Roads (3,522,834), Houston (3,453,220), Long Beach (9,3
84,368), Los Angeles (10,677,610), New York/New Jersey (8,985,929), Oakland (2,448,
243), Savannah (5,613,163), Seattle/Tacoma (3,736,206) (2021)LNG terminal(s) (expor
t):Calcasieu Pass (LA), Cameron (LA), Corpus Christi (TX), Cove Point (MD), Elba Is
land (GA), Freeport (TX), Sabine Pass (LA)note - two additional export facilities a
re under construction and expected to begin commercial operations in 2023-2024LNG t
erminal(s) (import):Cove Point (MD), Elba Island (GA), Everett (MA), Freeport (TX),
Golden Pass (TX), Hackberry (LA), Lake Charles (LA), Neptune (offshore), Northeast
Gateway (offshore), Pascagoula (MS), Sabine Pass (TX)river port(s):Baton Rouge, Pla
quemines, New Orleans (Mississippi River)cargo ports:Baton Rouge, Corpus Christi, H
ampton Roads, Houston, Long Beach, Los Angeles, New Orleans, New York, Plaquemines
(LA), Tampa, Texas Citycruise departure ports (passengers):Miami, Port Everglades,
Port Canaveral, Seattle, Long Beach Night view the port of Savannah, Georgia. Photo
courtesy of the US Coast Guard.:  View of the port of Charleston, South Carolina. P
hoto courtesy of the US Coast Guard.:  View of the port of Los Angeles, California.
Photo courtesy of US Coast Guard.:  View of the port of Long Beach, California. Pho
to courtesy of the US Coast Guard.:  An aerial view of the Golden Pass LNG export t
erminal at Port Arthur, Texas. Photo courtesy of US Coast Guard.:  A view of a liqu
ified natural gas (LNG) carrier docking at the Cove Point, Maryland LNG terminal. P
hoto courtesy of the US Coast Guard.:  The Marvel Crane, the first liquid natural g
as carrier to transport natural gas from the Cameron LNG facility in Louisiana. Pho
to courtesy of the US Coast Guard.: '
```

In [ ]:
```
#country_info = {}
#for country in name_list:
#    country_info[country] = get_info(country)
#    time.sleep(3)
```

In [ ]:
```
#import pickle

#with open("../Data/countries.pkl", "ab") as f:
#    pickle.dump(country_info, f)
```

In [ ]:
```
import pickle
country_info = {}
with open("../Data/countries.pkl", "rb") as f:
    while True:
        try:
            a = pickle.load(f)
        except EOFError:
            break
        else:
            country_info.update(a)
```

In [ ]:

**(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 [ ]:
```
def ports(country, info = country_info):
    if info[country].get('Ports and terminals'):
        given_string = info[country].get('Ports and terminals')
        major_seaports_match = re.search(r"major seaport\(s\):(.*?)(?=[(])", given_
        if major_seaports_match:
            # Check if the pattern is found
            # Get the extracted major seaports
            major_seaports_text = major_seaports_match.group(1).strip()
            # Split the text into a list of major seaports
            major_seaports = [port.strip() for port in major_seaports_text.split(",
            return major_seaports
        major_seaports_match = re.search(r"major seaport\(s\):(.*)", given_string)
        if major_seaports_match:
            major_seaports_text = major_seaports_match.group(1).strip()
            # Split the text into a list of major seaports
            major_seaports = [port.strip() for port in major_seaports_text.split(",
            return major_seaports

    else:
        return None
```

**(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 [ ]:
```
ports_dict = {}
for country in name_list:
    ports_dict[country] = ports(country)
```

In [ ]:
```
ports_dict['panama']
```

Out[ ]:
```
['Balboa', 'Colon', 'Cristobalcontainer port']
```

In [ ]:

In [ ]:
```python
def filter(dictionary):
    filtered = {}
    for key, ports in dictionary.items():
        if dictionary[key]:
            filtered[key] = [item for item in ports if ":" not in item]
    return filtered
```

In [ ]:
```python
filtered_ports = filter(ports_dict)
```

In [ ]:
```python
import lxml
nom_url = "https://nominatim.openstreetmap.org/search"
coord_list = []
for country, cities in filtered_ports.items():
    if cities:
        for city in cities:
            params = {
                'q': f'{country}, {city}',
                'format':'xml',
                'polygon_kml':'1',
                'addressdetails':'1'
            }
            response = requests.get(nom_url, params=params)
            if response.status_code == 200:
                bs = BeautifulSoup(response.text, 'xml')
                if bs.find('place'):
                    lat = bs.find('place')['lat']
                    lon = bs.find('place')['lon']
                    coord = float(lat), float(lon)
                    coord_list.append(coord)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[16], line 13
      6 for city in cities:
      7     params = {
      8         'q': f'{country}, {city}',
      9         'format':'xml',
     10         'polygon_kml':'1',
     11         'addressdetails':'1'
     12     }
---> 13     response = requests.get(nom_url, params=params)
     14     if response.status_code == 200:
     15         bs = BeautifulSoup(response.text, 'xml')

File c:\Users\alanp\anaconda3\Lib\site-packages\requests\api.py:73, in get(url, par
ams, **kwargs)
     62 def get(url, params=None, **kwargs):
     63     r"""Sends a GET request.
     64 
     65     :param url: URL for the new :class:`Request` object.
   (...)
     70     :rtype: requests.Response
     71     """
---> 73     return request("get", url, params=params, **kwargs)

File c:\Users\alanp\anaconda3\Lib\site-packages\requests\api.py:59, in request(meth
od, url, **kwargs)
     55 # By using the 'with' statement we are sure the session is closed, thus we
     56 # avoid leaving sockets open which can trigger a ResourceWarning in some
     57 # cases, and look like a memory leak in others.
     58 with sessions.Session() as session:
---> 59     return session.request(method=method, url=url, **kwargs)

File c:\Users\alanp\anaconda3\Lib\site-packages\requests\sessions.py:589, in Sessio
n.request(self, method, url, params, data, headers, cookies, files, auth, timeout,
allow_redirects, proxies, hooks, stream, verify, cert, json)
    584 send_kwargs = {
    585     "timeout": timeout,
    586     "allow_redirects": allow_redirects,
    587 }
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp

File c:\Users\alanp\anaconda3\Lib\site-packages\requests\sessions.py:703, in Sessio
n.send(self, request, **kwargs)
    700 start = preferred_clock()
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
    705 # Total elapsed time of the request (approximately)
    706 elapsed = preferred_clock() - start

File c:\Users\alanp\anaconda3\Lib\site-packages\requests\adapters.py:486, in HTTPAd
apter.send(self, request, stream, timeout, verify, cert, proxies)
    483     timeout = TimeoutSauce(connect=timeout, read=timeout)
    485 try:
--> 486     resp = conn.urlopen(
    487         method=request.method,
    488         url=url,
    489         body=request.body,
    490         headers=request.headers,
    491         redirect=False,
    492         assert_same_host=False,
    493         preload_content=False,
    494         decode_content=False,
    495         retries=self.max_retries,
    496         timeout=timeout,
    497         chunked=chunked,
    498     )
    500 except (ProtocolError, OSError) as err:
    501     raise ConnectionError(err, request=request)

File c:\Users\alanp\anaconda3\Lib\site-packages\urllib3\connectionpool.py:714, in H
TTPConnectionPool.urlopen(self, method, url, body, headers, retries, redirect, asse
rt_same_host, timeout, pool_timeout, release_conn, chunked, body_pos, **response_k
w)
    711     self._prepare_proxy(conn)
    713 # Make the request on the httplib connection object.
--> 714 httplib_response = self._make_request(
    715     conn,
    716     method,
    717     url,
    718     timeout=timeout_obj,
    719     body=body,
    720     headers=headers,
    721     chunked=chunked,
    722 )
    724 # If we're going to release the connection in ``finally:``, then
    725 # the response doesn't need to know about the connection. Otherwise
    726 # it will also try to release it and we'll have a double-release
    727 # mess.
    728 response_conn = conn if not release_conn else None

File c:\Users\alanp\anaconda3\Lib\site-packages\urllib3\connectionpool.py:403, in H
TTPConnectionPool._make_request(self, conn, method, url, timeout, chunked, **httpli
b_request_kw)
    401 # Trigger any extra validation we need to do.
    402 try:
--> 403     self._validate_conn(conn)
    404 except (SocketTimeout, BaseSSLError) as e:
    405     # Py2 raises this as a BaseSSLError, Py3 raises it as socket timeout.
    406     self._raise_timeout(err=e, url=url, timeout_value=conn.timeout)

File c:\Users\alanp\anaconda3\Lib\site-packages\urllib3\connectionpool.py:1053, in
HTTPSConnectionPool._validate_conn(self, conn)
   1051 # Force connect early to allow us to validate the connection.
   1052 if not getattr(conn, "sock", None):  # AppEngine might not have  `.sock`
-> 1053     conn.connect()
   1055 if not conn.is_verified:
   1056     warnings.warn(
   1057         (
   1058             "Unverified HTTPS request is being made to host '%s'. "
   (...)
   1063         InsecureRequestWarning,
   1064     )

File c:\Users\alanp\anaconda3\Lib\site-packages\urllib3\connection.py:419, in HTTPS
```

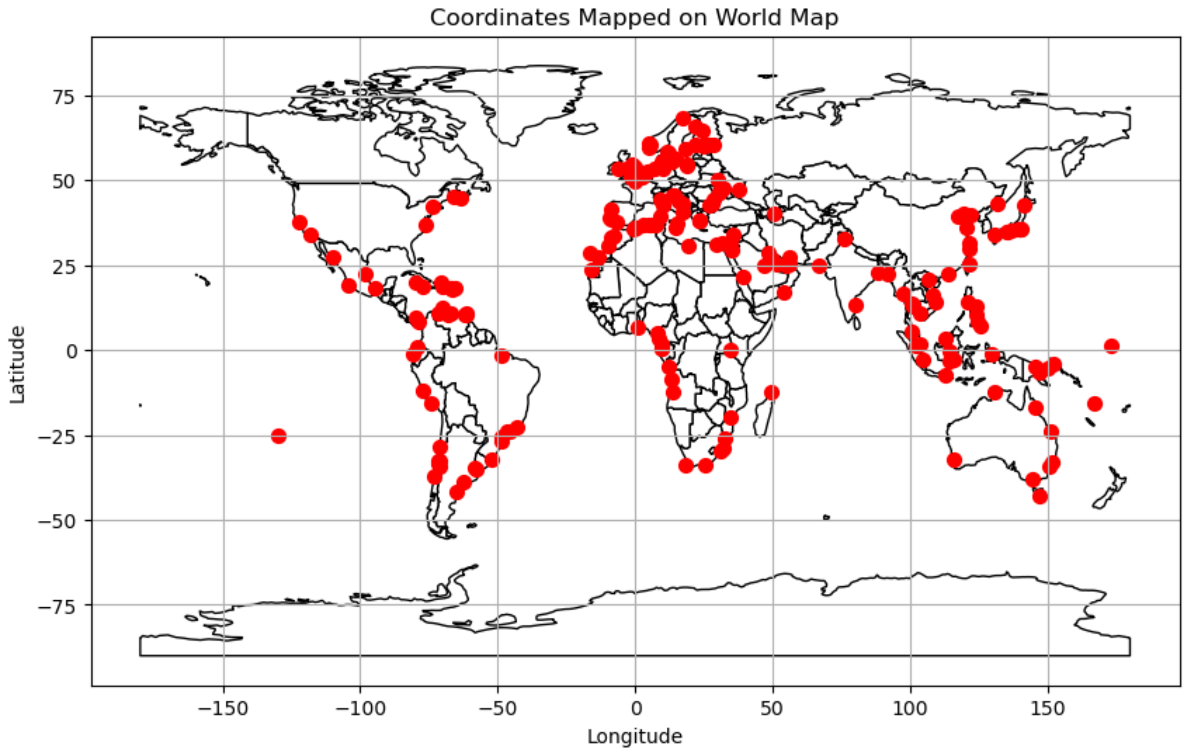
```
Connection.connect(self)
    410 if (
    411     not self.ca_certs
    412     and not self.ca_cert_dir
   (...)
    415     and hasattr(context, "load_default_certs")
    416 ):
    417     context.load_default_certs()
--> 419 self.sock = ssl_wrap_socket(
    420     sock=conn,
    421     keyfile=self.key_file,
    422     certfile=self.cert_file,
    423     key_password=self.key_password,
    424     ca_certs=self.ca_certs,
    425     ca_cert_dir=self.ca_cert_dir,
    426     ca_cert_data=self.ca_cert_data,
    427     server_hostname=server_hostname,
    428     ssl_context=context,
    429     tls_in_tls=tls_in_tls,
    430 )
    432 # If we're using all defaults and the connection
    433 # is TLSv1 or TLSv1.1 we throw a DeprecationWarning
    434 # for the host.
    435 if (
    436     default_ssl_context
    437     and self.ssl_version is None
    438     and hasattr(self.sock, "version")
    439     and self.sock.version() in {"TLSv1", "TLSv1.1"}
    440 ):

File c:\Users\alanp\anaconda3\Lib\site-packages\urllib3\util\ssl_.py:402, in ssl_wr
ap_socket(sock, keyfile, certfile, cert_reqs, ca_certs, server_hostname, ssl_versio
n, ciphers, ssl_context, ca_cert_dir, key_password, ca_cert_data, tls_in_tls)
    400 if ca_certs or ca_cert_dir or ca_cert_data:
    401     try:
--> 402         context.load_verify_locations(ca_certs, ca_cert_dir, ca_cert_data)
    403     except (IOError, OSError) as e:
    404         raise SSLError(e)

KeyboardInterrupt:
```

In [ ]:

```
import pickle
with open("../Data/coordinates.pkl", "rb") as f:
    coord_list = pickle.load(f)
```

```
In [ ]: import geopandas as gpd
        from shapely.geometry import Point

        # List of latitude and longitude coordinates

        # Create a GeoDataFrame from the coordinates
        geometry = [Point(lon, lat) for lat, lon in coord_list]
        gdf = gpd.GeoDataFrame(geometry=geometry, columns=['geometry'])

        # Plot the world map
        world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
        ax = world.plot(figsize=(10, 6), color='white', edgecolor='black')

        # Plot the coordinates on the map
        gdf.plot(ax=ax, color='red', markersize=50)

        # Show the plot
        ax.set_title('Coordinates Mapped on World Map')
        ax.set_xlabel('Longitude')
        ax.set_ylabel('Latitude')
        ax.grid(True)
        ax.set_aspect('auto')
```

```
C:\Users\alanp\AppData\Local\Temp\ipykernel_54444\1452157452.py:11: FutureWarning:
The geopandas.dataset module is deprecated and will be removed in GeoPandas 1.0. Yo
u can get the original 'naturalearth_lowres' data from https://www.naturalearthdat
a.com/downloads/110m-cultural-vectors/.
  world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
```



There are two points out in the sea which is either a port on a small island or incorrect coordinates, in addition, there is on point that may be inside a landlocked area

**(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 [ ]:
```
seaport_countries = []
for keys, cities in filtered_ports.items():
    if cities:
        seaport_countries.append(keys)
```

In [ ]:

In [ ]:
```
sea_port_bulk = {}
for country in seaport_countries:
    if country_info[country].get('Merchant marine'):
        sea_port_bulk[country] = country_info[country]['Merchant marine']
```

In [ ]:
```
import pandas as pd
num_bulk = {}
for country, string in sea_port_bulk.items():
    pattern = r'bulk carrier (\d{1,4}(?:,\d{3})*)'

    # Use re.search to find the first occurrence of the pattern in the string
    match = re.search(pattern, string)

    # Extract the number of bulk carriers from the matched group
    if match:
        num_bulk[country] = int(match.group(1).replace(',', ''))  # Remove commas f

bulk_df = pd.DataFrame(list(num_bulk.items()), columns = ['Country', 'Bulk Carriers
bulk_df.sort_values(by='Bulk Carriers', ascending=False).head(4)
```

Out[ ]:

| | Country | Bulk Carriers |
|---|---|---|
| **53** | panama | 2732 |
| **8** | marshall-islands | 1939 |
| **2** | liberia | 1895 |
| **28** | china | 1831 |

In [ ]:

Out[ ]:
```
'total:8,174 (2023)by type:bulk carrier 2732, container ship 671, general cargo 1,4
28, oil tanker 866, other 2,477 comparison ranking:total 3'
```

In [ ]:
```
landlocked_list = []

for key in country_info:
    if country_info[key].get('Coastline'):
        if 'landlocked' in country_info[key]['Coastline']:
            landlocked_list.append(key)
```

In [ ]:
```
land_marine = {}
for country in landlocked_list:
    if country_info[country].get('Merchant marine'):
        string = country_info[country]['Merchant marine']
        pattern = r'total:(\d{1,}(?:,\d{3})*)'
        # Use re.search to find the first occurrence of the pattern in the string
        match = re.search(pattern, string)

        # Extract the number after "total" from the matched group
        if match:
            total_number = int(match.group(1).replace(',', ''))
            land_marine[country] = total_number

marine_df = pd.DataFrame(list(land_marine.items()), columns = ['Country', 'Merchant
marine_df.sort_values(by='Merchant marine', ascending=False).head(4)
```

Out[ ]:

| | Country | Merchant marine |
|---|---|---|
| **11** | mongolia | 318 |
| **1** | azerbaijan | 312 |
| **9** | luxembourg | 147 |
| **6** | 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 [ ]:
```
def was_british(country, info = country_info):
    text = info[country]['Background']
    british_words = ["british", "great britain", "the united kingdom (uk)", "the br
    # Convert the text to lowercase for case-insensitive matching
    lowercase_text = text.lower()

    # Check if any word from the British, control, and separated lists exist in the
    return any(word in lowercase_text for word in british_words)
```

In [ ]:
```
print(was_british('pakistan'))
print(was_british('russia'))
```

```
True
False
```

In [ ]:
```
print(was_british('spain'))
print(was_british('united-states'))
```

```
False
True
```

In [ ]:

```
brit = {}
for country in country_info:
    if country_info[country].get('Background'):
        brit[country] = was_british(country)


brit_df = pd.DataFrame(list(brit.items()), columns = ['Name', 'was_british'])
brit_df.head()
```

Out[ ]:

|   | Name | was_british |
|---|---|---|
| **0** | afghanistan | True |
| **1** | akrotiri | False |
| **2** | albania | False |
| **3** | algeria | False |
| **4** | american-samoa | False |

**(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 [ ]:

```
iso = pd.read_csv('../Data/Country Data Codes.csv')

iso['Name'] = iso['Name'].apply(process_names)
full_info = pd.merge(iso, brit_df, on='Name', how = 'inner')
full_info
```

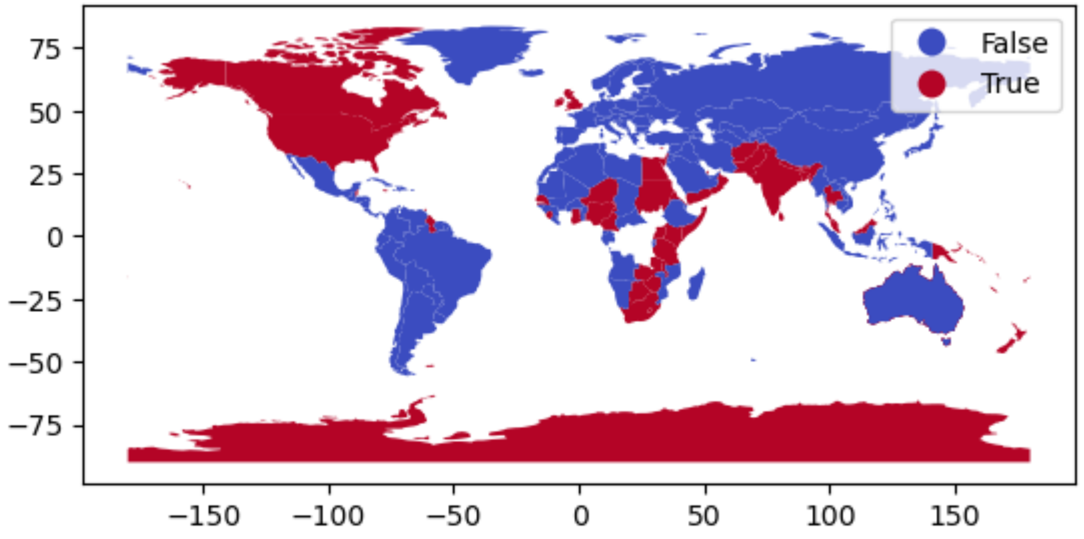
Out[ ]:

| | Name | GENC | ISO 3166 | Stanag | Internet | Comment | was_british |
|---|---|---|---|---|---|---|---|
| **0** | afghanistan | AFG | AF\|AFG\|004 | AFG | .af | NaN | True |
| **1** | akrotiri | XQZ | - | - | - | NaN | False |
| **2** | albania | ALB | AL\|ALB\|008 | ALB | .al | NaN | False |
| **3** | algeria | DZA | DZ\|DZA\|012 | DZA | .dz | NaN | False |
| **4** | american-samoa | ASM | AS\|ASM\|016 | ASM | .as | NaN | False |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **238** | west-bank | XWB | PS\|PSE\|275 | PSE | .ps | ISO identifies as Occupied Palestinian Territory | True |
| **239** | world | - | - | - | - | the Factbook uses the W data code from DIAM 65... | False |
| **240** | yemen | YEM | YE\|YEM\|887 | YEM | .ye | NaN | True |
| **241** | zambia | ZMB | ZM\|ZMB\|894 | ZMB | .zm | NaN | True |
| **242** | zimbabwe | ZWE | ZW\|ZWE\|716 | ZWE | .zw | NaN | True |

243 rows × 7 columns

In [ ]:

```python
from matplotlib import pyplot as plt
world_color = world.merge(full_info, how='left', left_on='iso_a3', right_on='Stanag
world_color.plot(column = 'was_british', cmap='coolwarm', legend=True)
plt.show()
```



**(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 [ ]:
```
def get_threshold(country, info=country_info):
        if info[country].get('Current health expenditure'):
                string = info[country]['Current health expenditure']

                pattern = r'(\d+(\.\d+)?)'

        # Use re.search to find the first occurrence of the pattern in the string
                match = re.search(pattern, string)

        # Extract the number from the matched group
                if match:
                    number = float(match.group(1))
                    return number
                else:
                       return 0
        else:
                       return 0
```

In [ ]:
```python
def get_pop_age(country, info = country_info):
    if info[country].get('Age structure'):
        string = info[country]['Age structure']
        age_pattern = r'(\d+-\d+ years|65 years and over)'
        population_pattern = r'male (\d+(?:,\d+)*)/female (\d+(?:,\d+)*)'
# Find all occurrences of age group, male, and female populations
        age_groups = re.findall(age_pattern, string)
        populations = re.findall(population_pattern, string)

# Print the results
        if populations:
            x = {}
            for i in range(len(age_groups)):
                pop = [int(populations[i][0].replace(",", "")), int(populations[i][
                x[age_groups[i]] = pop
            return x
        else:
            groups = ['0-14 years', '15-64 years', '65 years and over']
            x = {}
            for group in groups:
                x[group] = [0, 0]
            return x
    else:
            groups = ['0-14 years', '15-64 years', '65 years and over']
            x = {}
            for group in groups:
                x[group] = [0, 0]
            return x
```

In [ ]:
```python
thresholds = []
for country in country_info:
    thresholds.append(get_threshold(country))
len(set(thresholds))
```

Out[ ]:
```
92
```

In [ ]:
```python
get_pop_age('united-states')
```

Out[ ]:
```
{'0-14 years': [31509186, 30154408],
 '15-64 years': [108346275, 108100830],
 '65 years and over': [27589149, 33965270]}
```

In [ ]:
```python
pop_threshold = {}
for country in country_info:
    threshold = {}
    threshold['threshold'] = get_threshold(country)
    pop_threshold[country] = threshold
    population = {}
    population['population'] = get_pop_age(country)
    pop_threshold[country].update(population)
```

In [ ]:
```
data = pop_threshold['afghanistan']

# Convert dictionary to DataFrame
df = pd.DataFrame.from_dict(data['population'], orient='index', columns=['male_popu

# Add threshold column
df['threshold'] = data['threshold']

# Reset index to make 'year_group' a regular column
df.reset_index(inplace=True)
df.rename(columns={'index': 'year_group'}, inplace=True)
df
```

Out[ ]:

| | year_group | male_population | female_population | threshold |
|---|---|---|---|---|
| **0** | 0-14 years | 7926748 | 7686979 | 15.5 |
| **1** | 15-64 years | 11413654 | 11084665 | 15.5 |
| **2** | 65 years and over | 515147 | 604810 | 15.5 |

In [ ]:
```
df_list = []

# Iterate over the dictionaries and convert them to DataFrames
for data in pop_threshold.values():
    temp_df = pd.DataFrame.from_dict(data['population'], orient='index', columns=['
    temp_df['threshold'] = data['threshold']
    temp_df.reset_index(inplace=True)
    temp_df.rename(columns={'index': 'year_group'}, inplace=True)
    df_list.append(temp_df)

final_df = pd.concat(df_list, ignore_index=True)
aggregated = final_df.groupby(['threshold', 'year_group']).sum().reset_index()
aggregated
```

Out[ ]:

| | threshold | year_group | male_population | female_population |
|---|---|---|---|---|
| **0** | 0.0 | 0-14 years | 1060222707 | 998460689 |
| **1** | 0.0 | 15-64 years | 2683881185 | 2620523041 |
| **2** | 0.0 | 65 years and over | 403380240 | 505978931 |
| **3** | 1.7 | 0-14 years | 1522 | 1448 |
| **4** | 1.7 | 15-64 years | 8638 | 8558 |
| **...** | ... | ... | ... | ... |
| **271** | 18.8 | 15-64 years | 108346275 | 108100830 |
| **272** | 18.8 | 65 years and over | 27589149 | 33965270 |
| **273** | 21.5 | 0-14 years | 1745 | 1662 |
| **274** | 21.5 | 15-64 years | 3703 | 3664 |
| **275** | 21.5 | 65 years and over | 318 | 547 |

276 rows × 4 columns

In [ ]:
```
aggregated = aggregated.sort_values(by='threshold')

# Initialize cumulative population
cumulative_male_population = []
cumulative_female_population = []

# Iterate over each row
for index, row in aggregated.iterrows():
    # Filter DataFrame for the same year group and threshold condition
    filtered_df = aggregated[(aggregated['year_group'] == row['year_group']) & (agg

    # Calculate cumulative population
    cumulative_male_population.append(filtered_df['male_population'].sum())
    cumulative_female_population.append(filtered_df['female_population'].sum())

# Add cumulative population columns to DataFrame
aggregated['cumulative_male_population'] = cumulative_male_population
aggregated['cumulative_female_population'] = cumulative_female_population

final_aggregated = aggregated.drop(['male_population', 'female_population'], axis=1
final_aggregated['cumulative_female_population'] = final_aggregated['cumulative_fem
final_aggregated['cumulative_male_population'] = final_aggregated['cumulative_male_
```

In [ ]:
```
final_aggregated.head()
```

Out[ ]:

| | threshold | year_group | cumulative_male_population | cumulative_female_population |
|---|---|---|---|---|
| **0** | 0.0 | 0-14 years | 2.023938 | -1.903780 |
| **1** | 0.0 | 15-64 years | 5.220642 | -5.095570 |
| **2** | 0.0 | 65 years and over | 0.753471 | -0.939885 |
| **3** | 1.7 | 0-14 years | 0.963715 | -0.905319 |
| **4** | 1.7 | 15-64 years | 2.536761 | -2.475047 |

In [ ]:
```
final_aggregated.to_csv('../Data/aggregated.csv', index=False)
```

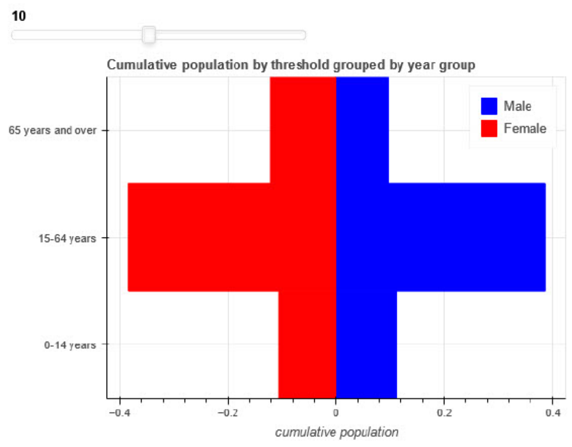
**(ii)** Using `bokeh.io` , create a client-based interactive opulation pyramid that displays the data from (i) according to a set threshold (or the closest threshold that exists). Make sure that the pyramid is well crafted, similar to this, but with a slider and only three population groups.

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

In [ ]:
```
from bokeh.palettes import DarkText, Vibrant3 as colors
import numpy as np
import pandas as pd
import copy
import bokeh.layouts
from bokeh.models import ColumnDataSource, CustomJS, CDSView, GroupFilter
from bokeh.models.widgets import Slider
from bokeh.plotting import figure, curdoc
bokeh.io.output_notebook()

final_aggregated = pd.read_csv('../Data/aggregated.csv')
source = ColumnDataSource(final_aggregated)
l = final_aggregated['year_group'].unique()

p = figure(y_range=l, title="Cumulative population by threshold grouped by year gro
p.hbar(y='year_group', right='cumulative_male_population', color=colors[0], source=
p.hbar(y='year_group', right='cumulative_female_population', color=colors[1], sourc

start = final_aggregated["threshold"].min()
end = final_aggregated["threshold"].max()

slider = bokeh.models.Slider(start = start, end = end, step = .2, value = start)

ts = final_aggregated['threshold'].unique()
def callback(attr, old, new):
    idx = np.absolute(ts - slider.value).argmin()
    value = ts[idx]

    new_ts = final_aggregated['threshold'] == value
    final_ts = final_aggregated[new_ts]
    new_source = ColumnDataSource(final_ts)
    source.data = dict(new_source.data)

slider.on_change("value", callback)

layout = bokeh.layouts.column(slider, p)
curdoc().add_root(layout)
```

BokehJS 3.2.1 successfully loaded.

In [ ]:
```
from IPython.display import display, Image
path = "../Images/Bokeh_Application.jpg"
display(Image(path, width=400, height=400))
```


10
Cumulative population by threshold grouped by year group
Male
Female
65 years and over
15-64 years
0-14 years
−0.4
−0.2
0
0.2
0.4
cumulative population