04-homework

March 9, 2025

1 STA 141B FQ 25 Homework Assignment 4

1.1 Instructions

- Complete the exercises below. Create more code chunks if necessary. Answer all questions. Show results for both the *test* and *run* cases.
- Export the Jupyter Notebook as an PDF file.
- Submit the PDF by **Sunday, March 9th, at 11:59 PM PT** to Gradescope.
- For each exercise, indicate the region of your answer in the PDF to facilitate grading.

1.2 Additional information

- Complete this worksheet yourself.
- You may use the internet or discuss possible approaches to solve the problems with other students. You are not allowed to share your code or your answers with other students.
- No other libraries than those explicitly allowed can be used.
- 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.
- Late homework submissions will not be accepted. No submissions will be accepted by email.
- The total number of points for this assignment is 20.

Exercise 1

Lets obtain movie information for the movies available on the Internet Movie Script Database IMSDb.

(a) Use the *Alphabetical* section to obtain the URL of all movies. How many different movies do you obtain?

[216]:
```
from concurrent.futures import ThreadPoolExecutor

import pandas as pd
import requests
from bs4 import BeautifulSoup

# Base URL of IMSDb
base_url = "https://imsdb.com"
```

[2]:
```
alphabet = ['0', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',␣
↪'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
```

[7]:
```
URLs = []

for letter in alphabet:
  url = f"{base_url}/alphabetical/{letter}"
  response = requests.get(url)
  soup = BeautifulSoup(response.text, 'html.parser')
  tables = soup.find_all("table")
  assert(len(tables) > 1)
  second_table = tables[1]
  data_cells = second_table.find_all("td")
  assert(len(data_cells) > 2)
  third_data_cell = data_cells[0]
  ps = third_data_cell.find_all("p")
  for p in ps:
    a = p.find("a")
    URLs.append(a["href"])
```

[12]:
```
df = pd.DataFrame(URLs, columns=["URL"])
df = df.drop_duplicates()
df.head()
```

[12]:

	URL
0	/Movie Scripts/10 Things I Hate About You Scri…
1	/Movie Scripts/12 Script.html
2	/Movie Scripts/12 and Holding Script.html
3	/Movie Scripts/12 Monkeys Script.html
4	/Movie Scripts/12 Years a Slave Script.html

[13]:
```
f'There are {df.shape[0]} movies'
```

[13]:
```
'There are 1295 movies'
```

[14]:
```
df.to_csv("movies.csv", index=False)
```

[211]:
```
df = pd.read_csv("movies.csv")
```

[212]:
```
df.head()
```

[212]:

	URL
0	/Movie Scripts/10 Things I Hate About You Scri…
1	/Movie Scripts/12 Script.html
2	/Movie Scripts/12 and Holding Script.html
3	/Movie Scripts/12 Monkeys Script.html
4	/Movie Scripts/12 Years a Slave Script.html

(b) For every movie, obtain the title, writers, genres, script date and movie release date.

Test:

```
> get_movie_details('/Movie Scripts/Feast Script.html')
('Feast',
 {'writers': ['Patrick Melton', 'Marcus Dunston'],
  'genres': ['Action', 'Comedy', 'Horror', 'Thriller'],
  'script_date': 2004,
  'release_date': 2006})
```

(i) Which movie has the greatest observed distance between script and movie release date? **(ii)** Which writer has written the most movies?

[5]:
```
df = pd.read_csv("movies.csv")
```

[215]:
```
def get_movie_details(endpoint):
  title = endpoint.split("/")[-1][:-4]

  with requests.Session() as session:
    response = session.get(f'{base_url}{endpoint}')

  soup = BeautifulSoup(response.text, 'html.parser')

  writers = [*map(
    lambda x: x.text,
    soup.find_all('a', href=lambda x: x and x.startswith('/writer.php?')),
  )]

  genres = [*map(
    lambda x: x.text,
    soup.find_all('a', href=lambda x: x and x.startswith('/genre/')),
  )]

  script_date_tag = soup.find('b', string='Script Date')
  script_date = script_date_tag.find_next(string=True).find_next(string=True).
↪strip()[2:] if script_date_tag else None
  release_date_tag = soup.find('b', string='Movie Release Date')
  release_date = release_date_tag.find_next(string=True).find_next(string=True).
↪strip()[2:] if release_date_tag else None

  return title, {
    'writers': writers,
    'genres': genres,
    'script_date': script_date,
    'release_date': release_date,
  }

get_movie_details("/Movie Scripts/Feast Script.html")
```

[215]:
```
('Feast Script.',
 {'writers': ['Patrick Melton', 'Marcus Dunston'],
```

```
 'genres': ['Action',
  'Adventure',
  'Animation',
  'Comedy',
  'Crime',
  'Drama',
  'Family',
  'Fantasy',
  'Film-Noir',
  'Horror',
  'Musical',
  'Mystery',
  'Romance',
  'Sci-Fi',
  'Short',
  'Thriller',
  'War',
  'Western',
  'Action',
  'Comedy',
  'Horror',
  'Thriller'],
 'script_date': 'May 2004',
 'release_date': 'September 2006'})
```

[217]:
```
with ThreadPoolExecutor() as executor:
    results = list(executor.map(get_movie_details, df['URL']))

# Convert results into a DataFrame
df[['title', 'details']] = pd.DataFrame(results, index=df.index)
```

[218]:
```
df.to_csv("movies.csv", index=False)
```

[219]:
```
df.head()
```

[219]:

	URL	title
0	/Movie Scripts/10 Things I Hate About You Scri…	10 Things I Hate About You Script.
1	/Movie Scripts/12 Script.html	12 Script.
2	/Movie Scripts/12 and Holding Script.html	12 and Holding Script.
3	/Movie Scripts/12 Monkeys Script.html	12 Monkeys Script.
4	/Movie Scripts/12 Years a Slave Script.html	

```
4            12 Years a Slave Script.

                                                 details
0  {'writers': ['Karen McCullah Lutz', 'Kirsten S…
1  {'writers': ['Lawrence Bridges'], 'genres': ['…
2  {'writers': ['Anthony Cipriano'], 'genres': ['…
3  {'writers': ['David Peoples', 'Janet Peoples']…
4  {'writers': ['John Ridley'], 'genres': ['Actio…
```

[220]:
```python
# Function to compute the difference in days
def date_diff(details):
    try:
        script_date = pd.to_datetime(details.get('script_date'))
        release_date = pd.to_datetime(details.get('release_date'))
        return abs((release_date - script_date).days) if pd.notna(script_date)␣
 ↪and pd.notna(release_date) else None
    except:
        return None

# Compute differences
df['date_diff'] = df['details'].apply(date_diff)

# Find the row with the max difference
max_diff_row = df.loc[df['date_diff'].idxmax()]
```

[223]:
```python
f'The {max_diff_row['title'][:-1]} has the greatest observed distance of␣
 ↪{max_diff_row['date_diff']} between script date and release date'
```

[223]:
```
'The Ricky Stanicky Script has the greatest observed distance of 5234.0 between
script date and release date'
```

[224]:
```python
from collections import Counter

# Function to extract writers list
def get_writers(details):
    return details.get('writers', []) if isinstance(details, dict) else []

# Extract and flatten the list of all writers
all_writers = df['details'].apply(get_writers).explode()

# Count occurrences of each writer
writer_counts = Counter(all_writers.dropna())

# Find the writer with the most movies
most_common_writer, most_movies = writer_counts.most_common(1)[0]
```

[226]:
```
f"The writer with the most movies is {most_common_writer} with {most_movies}␣
↪movies."
```

[226]:
```
'The writer with the most movies is Ethan Coen with 15 movies.'
```

Exercise 2

(a, i) Lets retrieve data from the CIA World Factbook. Using devtools, find a way to retrieve the names of all listed world entities. *How many distinct world entities did you find? (Hint: I found more than 228 and less than 261)*

(ii) In order to navigate to their respective site, I assembled the path by processing the country names. Retrieve all country specific data in JSON format.

[141]:
```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.select import Select

import pandas as pd
import requests
import json

from bs4 import BeautifulSoup
from time import sleep
```

[142]:
```
# (i)
driver = webdriver.Chrome()
driver.get("https://www.cia.gov/the-world-factbook/countries/")
select_num_pages = Select(driver.find_element(By.CSS_SELECTOR, "select.
↪per-page"))
```

[143]:
```
select_num_pages.select_by_visible_text("All")
sleep(1)
```

[144]:
```
links = driver.find_elements(By.XPATH, "//a[starts-with(@href, '/
↪the-world-factbook/countries/')]")
```

[145]:
```
link_texts = map(lambda link: link.text, links)
```

[146]:
```
link_urls = map(lambda link: link.get_attribute("href"), links)
```

[147]:
```
df = pd.DataFrame({"countries": link_texts, "urls": link_urls})
```

[148]:
```
driver.quit()
```

[149]:
```
df = df[df["countries"] != "Countries"]
```

[150]:
```
def get_country_json(url):
  country = url.split("/")[-2]
```

```
    response = requests.get(f'https://www.cia.gov/the-world-factbook/page-data/
    ↪countries/{country}/page-data.json')
    return response.text if response.status_code == 200 else None
```

[151]:
```
from concurrent.futures import ThreadPoolExecutor

with ThreadPoolExecutor() as executor:
    df['json_data'] = [*executor.map(get_country_json, df['urls'])]

df.head()
```

[151]:

	countries	urls	\
1	Afghanistan	https://www.cia.gov/the-world-factbook/countri…	
2	Akrotiri and Dhekelia	https://www.cia.gov/the-world-factbook/countri…	
3	Albania	https://www.cia.gov/the-world-factbook/countri…	
4	Algeria	https://www.cia.gov/the-world-factbook/countri…	
5	American Samoa	https://www.cia.gov/the-world-factbook/countri…	

	json_data
1	{"componentChunkName":"component---src-templat…
2	{"componentChunkName":"component---src-templat…
3	{"componentChunkName":"component---src-templat…
4	{"componentChunkName":"component---src-templat…
5	{"componentChunkName":"component---src-templat…

[152]:
```
df.to_csv("countries.csv", index=False)
```

[228]:
```
df = pd.read_csv("countries.csv")
```

[153]:
```
f'There are {df['urls'].nunique()} unique entities in the world'
```

[153]:
```
'There are 254 unique entities in the world'
```

(b, i) We are interested in the key ports of each country. Write a function `get_port_data` that takes the country and returns all key ports in a list. *How many ports in total did you find?*

Test:

```
>get_port_data('Afghanistan')

>get_port_data('Algeria')
['Alger',
 'Annaba',
 'Arzew',
 'Arzew El Djedid',
 'Bejaia',
 'Mers El Kebir',
 'Oran',
 'Port Methanier',
```

```
 'Skikda']

>get_port_data('United States')
['Baltimore',
 'Boston',
 'Brooklyn',
 'Buffalo',
 'Chester',
 'Cleveland',
 'Detroit',
 'Galveston',
 'Houston',
 'Los Angeles',
 'Louisiana Offshore Oil Port (LOOP)',
 'Mobile',
 'New Orleans',
 'New York City',
 'Norfolk',
 'Oakland',
 'Philadelphia',
 'Portland',
 'San Francisco',
 'Seattle',
 'Tri-City Port']
```

[154]:
```
def get_port_data(country):
  data = json.loads(df.loc[df['countries'] == country, 'json_data'].iloc[0])
  ports_object = next((item for item in␣
 ↪data['result']['data']['fields']['nodes'] if item.get('name') == "Ports"),␣
 ↪None)
  if ports_object is None: return None
  soup = BeautifulSoup(ports_object['data'], 'html.parser')
  strong_tag = soup.find('strong', string="key ports:")
  return strong_tag.next_sibling.strip().split(', ')
```

[155]:
```
get_port_data('Afghanistan')
```

[156]:
```
get_port_data('Algeria')
```

[156]:
```
['Alger',
 'Annaba',
 'Arzew',
 'Arzew El Djedid',
 'Bejaia',
 'Mers El Kebir',
 'Oran',
 'Port Methanier',
```

```
 'Skikda']
```

[158]:
```
get_port_data('United States')
```

[158]:
```
['Baltimore',
 'Boston',
 'Brooklyn',
 'Buffalo',
 'Chester',
 'Cleveland',
 'Detroit',
 'Galveston',
 'Houston',
 'Los Angeles',
 'Louisiana Offshore Oil Port (LOOP)',
 'Mobile',
 'New Orleans',
 'New York City',
 'Norfolk',
 'Oakland',
 'Philadelphia',
 'Portland',
 'San Francisco',
 'Seattle',
 'Tri-City Port']
```

[159]:
```
df['ports'] = df['countries'].map(get_port_data)
```

[160]:
```
df.drop(columns=['json_data', 'urls'], inplace=True)
df.head()
```

[160]:

	countries	ports
1	Afghanistan	None
2	Akrotiri and Dhekelia	None
3	Albania	[Durres, Shengjin, Vlores]
4	Algeria	[Alger, Annaba, Arzew, Arzew El Djedid, Bejaia…
5	American Samoa	[Pago Pago Harbor]

(ii) Where are these ports? Use the Nominatim API to obtain latitude-longitude pairs for each port. *How many pairs did you find?*

[166]:
```
# TODO: this and everything below
ports = df.explode('ports').dropna(subset=['ports'])
```

[167]:
```
ports.head()
```

[167]:

	countries	ports
3	Albania	Durres

```
3   Albania  Shengjin
3   Albania    Vlores
4   Algeria     Alger
4   Algeria    Annaba
```

[168]:
```
ports.to_csv("ports.csv", index=False)
```

[169]:
```
ports = pd.read_csv("ports.csv")
```

[333]:
```
def get_port_info(port, country):
  country = country.split(', ')
  if len(country) == 2:
    country[0], country[1] = country[1], country[0]
  country = ' '.join(country)
  port = port.replace(' ', '%20')
  url = f'https://nominatim.openstreetmap.org/search?
 ↪q={port}%20{country}&polygon_geojson=1&format=jsonv2'
  headers = {
    'User-Agent': 'MyWebScraper (ajowe@ucdavis.edu)'
  }
  response = requests.get(url, headers=headers)
  sleep(1)
  return response.json()
```

[336]:
```
ports['info'] = ports.apply(lambda row: get_port_info(row['ports'],␣
 ↪row['countries']), axis=1)
```

[337]:
```
ports.to_csv("ports.csv", index=False)
```

(iii) Add markers to a world map identifying each found port. The result should look something like this:

[338]:
```
ports.head()
```

[338]:
```
  countries     ports                                                info  \
0   Albania    Durres  [{'place_id': 51158903, 'licence': 'Data © Ope…
1   Albania  Shengjin  [{'place_id': 50678288, 'licence': 'Data © Ope…
2   Albania    Vlores  [{'place_id': 51200114, 'licence': 'Data © Ope…
3   Algeria     Alger  [{'place_id': 46754164, 'licence': 'Data © Ope…
4   Algeria    Annaba  [{'place_id': 47732893, 'licence': 'Data © Ope…

                          coords
0          (41.313255, 19.4462348)
1         (41.8097827, 19.5989755)
2  (40.154212, 19.752180995095102)
3          (36.7729333, 3.0588445)
4          (36.8982165, 7.7549272)
```

[339]:
```
def port_to_coords(info):
  if len(info) == 0: return None
  return info[0]['lat'], info[0]['lon']

ports['coords'] = ports['info'].apply(port_to_coords)
```

[343]:
```
coords = ports['coords']
coords = coords.dropna()
```

[342]:
```
import plotly.express as px

lats, longs = zip(*coords)  # Extract latitudes and longitudes

# Create a DataFrame for Plotly
df = pd.DataFrame({'Latitude': lats, 'Longitude': longs})

# Plot using Plotly Express
fig = px.scatter_geo(df,
                     lat='Latitude',
                     lon='Longitude',
                     projection="natural earth",
                     title="World Map with Coordinates",
                     color_discrete_sequence=['blue'])  # Make points blue

# Show the map
fig.show()
```

(c) Lets learn about each nations merchant marine! Write a function `merchant_marine` that takes the country and returns the merchant marine as follows:

```
>merchant_marine('Angola')
{'bulk carrier': 0,
 'container ship': 0,
 'general cargo': 13,
 'oil tanker': 8,
 'other': 43}
```

List the five most merchant marines measured **(i)** by the total amount of ships, **(ii)** by the total amount of *non-other* ships **(iii)** and by the total amout of oil tankers.

[287]:
```
import re

def removeExtraNote(ships):
    return re.sub(r'\s*\([^)]*\)$', '', ships)

def merchant_marine(country):
  print(country)
  merchant_marine_json = json.loads(df.loc[df['countries'] == country,␣
 ↪'json_data'].squeeze())
```

```
  nodes = merchant_marine_json['result']['data']['fields']['nodes']
  fleet = next((item for item in nodes if item.get("name") == "Merchant␣
↪marine"), None)
  if fleet is None: return None
  soup = BeautifulSoup(fleet['data'], 'html.parser')
  strong_tag = soup.find('strong', string="by type:")
  if strong_tag is None: return None
  ships = removeExtraNote(strong_tag.next_sibling.strip())
  ships = ships.strip().split(', ')
  res = {}
  for ship in ships:
    tokens = ship.split(' ')
    name = ' '.join(tokens[:-1])
    quantity = int(tokens[-1].replace(',', ''))
    res[name] = quantity
  return res
```

[263]:
```
df_unique = df.drop_duplicates(subset='urls', keep='first')
```

[288]:
```
df_unique['merchant_marine'] = df_unique['countries'].apply(lambda x:␣
↪merchant_marine(x) if pd.notna(x) else None)
```

```
Afghanistan
Akrotiri and Dhekelia
Albania
Algeria
American Samoa
Andorra
Angola
Anguilla
Antarctica
Antigua and Barbuda
Argentina
Armenia
Aruba
Ashmore and Cartier Islands
Australia
Austria
Azerbaijan
Bahamas, The
Bahrain
Baker Island
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
```

Bermuda
Bhutan
Bolivia
Bosnia and Herzegovina
Botswana
Bouvet Island
Brazil
British Indian Ocean Territory
British Virgin Islands
Brunei
Bulgaria
Burkina Faso
Burma
Burundi
Cabo Verde
Cambodia
Cameroon
Canada
Cayman Islands
Central African Republic
Chad
Chile
China
Christmas Island
Clipperton Island
Cocos (Keeling) Islands
Colombia
Comoros
Congo, Democratic Republic of the
Congo, Republic of the
Cook Islands
Coral Sea Islands
Costa Rica
Cote d'Ivoire
Croatia
Cuba
Curacao
Cyprus
Czechia
Denmark
Djibouti
Dominica
Dominican Republic
Ecuador
Egypt
El Salvador
Equatorial Guinea
Eritrea

Estonia
Eswatini
Ethiopia
European Union
Falkland Islands (Islas Malvinas)
Faroe Islands
Fiji
Finland
France
French Polynesia
French Southern and Antarctic Lands
Gabon
Gambia, The
Gaza Strip
Georgia
Germany
Ghana
Gibraltar
Greece
Greenland
Grenada
Guam
Guatemala
Guernsey
Guinea
Guinea-Bissau
Guyana
Haiti
Heard Island and McDonald Islands
Holy See (Vatican City)
Honduras
Hong Kong
Hungary
Iceland
India
Indonesia
Iran
Iraq
Ireland
Isle of Man
Israel
Italy
Jamaica
Jan Mayen
Japan
Jersey
Jordan
Kazakhstan

Kenya
Kiribati
Korea, North
Korea, South
Kosovo
Kuwait
Kyrgyzstan
Laos
Latvia
Lebanon
Lesotho
Liberia
Libya
Liechtenstein
Lithuania
Luxembourg
Macau
Madagascar
Malawi
Malaysia
Maldives
Mali
Malta
Marshall Islands
Mauritania
Mauritius
Mexico
Micronesia, Federated States of
Moldova
Monaco
Mongolia
Montenegro
Montserrat
Morocco
Mozambique
Namibia
Nauru
Navassa Island
Nepal
Netherlands
New Caledonia
New Zealand
Nicaragua
Niger
Nigeria
Niue
Norfolk Island
North Macedonia

Northern Mariana Islands
Norway
Oman
Pakistan
Palau
Panama
Papua New Guinea
Paracel Islands
Paraguay
Peru
Philippines
Pitcairn Islands
Poland
Portugal
Puerto Rico
Qatar
Romania
Russia
Rwanda
Saint Barthelemy
Saint Helena, Ascension, and Tristan da Cunha
Saint Kitts and Nevis
Saint Lucia
Saint Martin
Saint Pierre and Miquelon
Saint Vincent and the Grenadines
Samoa
San Marino
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Sint Maarten
Slovakia
Slovenia
Solomon Islands
Somalia
South Africa
South Georgia and South Sandwich Islands
South Sudan
Spain
Spratly Islands
Sri Lanka
Sudan
Suriname

```
Svalbard
Sweden
Switzerland
Syria
Taiwan
Tajikistan
Tanzania
Thailand
Timor-Leste
Togo
Tokelau
Tonga
Trinidad and Tobago
Tunisia
Turkey (Turkiye)
Turkmenistan
Turks and Caicos Islands
Tuvalu
Uganda
Ukraine
United Arab Emirates
United Kingdom
United States
Uruguay
Uzbekistan
Vanuatu
Venezuela
Vietnam
Virgin Islands
Wake Island
Wallis and Futuna
West Bank
World
Yemen
Zambia
Zimbabwe
```

```
/var/folders/1y/4sm5vl2d7_xdd0jwk0tgnhn00000gn/T/ipykernel_2256/2214531112.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_unique['merchant_marine'] = df_unique['countries'].apply(lambda x:
merchant_marine(x) if pd.notna(x) else None)
```

[294]:
```
df_unique.drop(columns=['urls', 'json_data'], inplace=True)
```

```
/var/folders/1y/4sm5vl2d7_xdd0jwk0tgnhn00000gn/T/ipykernel_2256/3378057982.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_unique.drop(columns=['urls', 'json_data'], inplace=True)
```

[295]:
```
df_unique.head()
```

[295]:

	countries	merchant_marine
0	Afghanistan	None
1	Akrotiri and Dhekelia	None
2	Albania	{'general cargo': 46, 'oil tanker': 1, 'other'…
3	Algeria	{'bulk carrier': 1, 'container ship': 4, 'gene…
4	American Samoa	None

[298]:
```
# Initialize columns for calculations
df_unique["total_ships"] = 0
df_unique["non_other_ships"] = 0
df_unique["oil_tankers"] = 0

# Process each row
for index, row in df_unique.iterrows():
    if isinstance(row["merchant_marine"], dict):  # Ensure valid data exists
        ship_counts = row["merchant_marine"]  # Directly use dictionary

        # (i) Total ships
        df_unique.at[index, "total_ships"] = sum(ship_counts.values())

        # (ii) Non-other ships
        df_unique.at[index, "non_other_ships"] = sum(v for k, v in ship_counts.
 ↪items() if k != "other")

        # (iii) Oil tankers
        df_unique.at[index, "oil_tankers"] = ship_counts.get("oil tanker", 0)

# Get top 5 for each category
top_total_ships = df_unique.nlargest(5, "total_ships")[["countries",␣
 ↪"total_ships"]]
top_non_other_ships = df_unique.nlargest(5, "non_other_ships")[["countries",␣
 ↪"non_other_ships"]]
top_oil_tankers = df_unique.nlargest(5, "oil_tankers")[["countries",␣
 ↪"oil_tankers"]]

# Print results
print("Top 5 by Total Ships:\n", top_total_ships)
```

```
print("\nTop 5 by Non-Other Ships:\n", top_non_other_ships)
print("\nTop 5 by Oil Tankers:\n", top_oil_tankers)
```

```
Top 5 by Total Ships:
      countries  total_ships
257       World       103577
110   Indonesia        11422
48        China         8314
181      Panama         8174
119       Japan         5229

Top 5 by Non-Other Ships:
      countries  non_other_ships
257       World            50478
181      Panama             5697
48        China             4838
137     Liberia             4116
110   Indonesia             3440

Top 5 by Oil Tankers:
             countries  oil_tankers
257              World        11604
48               China         1196
149  Marshall Islands         1039
137           Liberia         1038
181            Panama          866
/var/folders/1y/4sm5vl2d7_xdd0jwk0tgnhn00000gn/T/ipykernel_2256/4154705169.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_unique["total_ships"] = 0
/var/folders/1y/4sm5vl2d7_xdd0jwk0tgnhn00000gn/T/ipykernel_2256/4154705169.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_unique["non_other_ships"] = 0
/var/folders/1y/4sm5vl2d7_xdd0jwk0tgnhn00000gn/T/ipykernel_2256/4154705169.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_unique["oil_tankers"] = 0
```

[ ]: