The US Geological Survey publishes a list of Strategic Minerals (https://www.usgs.gov/news/national-news-release/us-geological-survey-releases-2022-list-critical-minerals). Having a secure supply of these minerals is essential to our security and economic prosperity. However many of these minerals are sourced from outside of the US. This assignment is to develop a reference catalog of the source or sources of each of these minerals and a judgement on the reliability of each source under stressed circumstance (e.g. war, economic crisis, etc.)

Notes:

You will need to identify a source or sources for each of the minerals in the 2022 List of Critical Minerals

You will need to categorize each source country as an ally, a competitor or a neutral party.

You will need to develop data visualizations that tell the story of source dependency and shortfall impact.

This assignment is due at the end of week fourteen of the semester

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import requests
import json

import comtradeapicall as ct
key = pd.read_json('data/api_keys.json', typ='series')['uncomtrade']

from fuzzywuzzy import fuzz
from fuzzywuzzy import process

import geopandas as gpd
import plotly.graph_objects as go
import plotly.express as px

from utils import country_relationships
```

In [ ]:
```
full_query = False
```

List of Minerals

In [ ]:
```
minerals = {
    'Aluminium': 'used in almost all sectors of the economy',
    'Antimony': 'used in lead-acid batteries and flame retardants',
    'Arsenic': 'used in semi-conductors',
    'Barite': 'used in hydrocarbon production',
```

```
    'Beryllium': 'used as an alloying agent in aerospace and defense industries',
    'Bismuth': 'used in medical and atomic research',
    'Cerium': 'used in catalytic converters, ceramics, glass, metallurgy, and polis
    'Cesium': 'used in research and development',
    'Chromium': 'used primarily in stainless steel and other alloys',
    'Cobalt': 'used in rechargeable batteries and superalloys',
    'Dysprosium': 'used in permanent magnets, data storage devices, and lasers',
    'Erbium': 'used in fiber optics, optical amplifiers, lasers, and glass colorant
    'Europium': 'used in phosphors and nuclear control rods',
    'Fluorspar': 'used in the manufacture of aluminum, cement, steel, gasoline, and
    'Gadolinium': 'used in medical imaging, permanent magnets, and steelmaking',
    'Gallium': 'used for integrated circuits and optical devices like LEDs',
    'Germanium': 'used for fiber optics and night vision applications',
    'Graphite': 'used for lubricants, batteries, and fuel cells',
    'Hafnium': 'used for nuclear control rods, alloys, and high-temperature ceramic
    'Holmium': 'used in permanent magnets, nuclear control rods, and lasers',
    'Indium': 'used in liquid crystal display screens',
    'Iridium': 'used as coating of anodes for electrochemical processes and as a ch
    'Lanthanum': 'used to produce catalysts, ceramics, glass, polishing compounds,
    'Lithium': 'used for rechargeable batteries',
    'Lutetium': 'used in scintillators for medical imaging, electronics, and some c
    'Magnesium': 'used as an alloy and for reducing metals',
    'Manganese': 'used in steelmaking and batteries',
    'Neodymium': 'used in permanent magnets, rubber catalysts, and in medical and i
    'Nickel': 'used to make stainless steel, superalloys, and rechargeable batterie
    'Niobium': 'used mostly in steel and superalloys',
    'Palladium': 'used in catalytic converters and as a catalyst agent',
    'Platinum': 'used in catalytic converters',
    'Praseodymium': 'used in permanent magnets, batteries, aerospace alloys, cerami
    'Rhodium': 'used in catalytic converters, electrical components, and as a catal
    'Rubidium': 'used for research and development in electronics',
    'Ruthenium': 'used as catalysts, as well as electrical contacts and chip resist
    'Samarium': 'used in permanent magnets, as an absorber in nuclear reactors, and
    'Scandium': 'used for alloys, ceramics, and fuel cells',
    'Tantalum': 'used in electronic components, mostly capacitors and in superalloy
    'Tellurium': 'used in solar cells, thermoelectric devices, and as alloying addi
    'Terbium': 'used in permanent magnets, fiber optics, lasers, and solid-state de
    'Thulium': 'used in various metal alloys and in lasers',
    'Tin': 'used as protective coatings and alloys for steel',
    'Titanium': 'used as a white pigment or metal alloys',
    'Tungsten': 'primarily used to make wear-resistant metals',
    'Vanadium': 'primarily used as alloying agent for iron and steel',
    'Ytterbium': 'used for catalysts, scintillometers, lasers, and metallurgy',
    'Yttrium': 'used for ceramic, catalysts, lasers, metallurgy, and phosphors',
    'Zinc': 'primarily used in metallurgy to produce galvanized steel',
    'Zirconium': 'used in the high-temperature ceramics and corrosion-resistant all
}
```

Data

https://comtradeplus.un.org/

https://pypi.org/project/comtradeapicall/

In [ ]:

```
ref_df = ct.listReference()
url = ref_df.loc[ref_df.category == 'cmd:HS', 'fileuri'].values[0]
response = requests.get(url)
response_json = json.loads(response.text)

commodities_codes_df = pd.DataFrame(response_json['results'])

minerals_df = pd.DataFrame()

for mineral in minerals:
    temp_df = commodities_codes_df.loc[commodities_codes_df.text.str.contains(f'{mi
    temp_df['mineral'] = mineral
    if len(temp_df) == 0:
        print(f'{mineral} not found')
    else:
        minerals_df = pd.concat([minerals_df, temp_df])

minerals_df.head()
```

```
Barite not found
Cesium not found
Dysprosium not found
Erbium not found
Europium not found
Holmium not found
Lanthanum not found
Lutetium not found
Neodymium not found
Praseodymium not found
Rubidium not found
Samarium not found
Terbium not found
Thulium not found
Ytterbium not found
```

Out[ ]:

	id	text	parent	isLeaf	aggrLevel	standardUnitAbbr	mineral
1437	2510	2510 - Natural calcium phosphates; natural alu...	25	0	4	n/a	Aluminium
1438	251010	251010 - Natural calcium phosphates, natural a...	2510	1	6	kg	Aluminium
1439	251020	251020 - Natural calcium phosphates, natural a...	2510	1	6	kg	Aluminium
1534	2606	2606 - Aluminium ores and concentrates	26	0	4	n/a	Aluminium
1535	260600	260600 - Aluminium ores and concentrates	2606	1	6	kg	Aluminium

In [ ]:
```
url = ref_df.loc[ref_df.category == 'reporter', 'fileuri'].values[0]
response = requests.get(url)
response_json = json.loads(response.text)
reporter_df = pd.DataFrame(response_json['results'])
reporter_df.head()
```

Out[ ]:

	id	text	reporterCode	reporterDesc	reporterNote	reporterCodeIsoAlpha2	repc
0	4	Afghanistan	4	Afghanistan	Afghanistan	AF	
1	8	Albania	8	Albania	Albania	AL	
2	12	Algeria	12	Algeria	Algeria	DZ	
3	20	Andorra	20	Andorra	Andorra	AD	
4	24	Angola	24	Angola	Angola	AO	

In [ ]:
```
url = ref_df.loc[ref_df.category == 'partner', 'fileuri'].values[0]
response = requests.get(url)
response_json = json.loads(response.text)
```

```
partner_df = pd.DataFrame(response_json['results'])
print(f"Partner code for the world: {partner_df.loc[partner_df.text == 'World', 'Pa
```

```
Partner code for the world: 0
```

Query

In [ ]:

```
if full_query:
    trade_df = pd.DataFrame(index = [x for x in reporter_df.reporterCode.unique()])
    minerals = [x for x in minerals_df.mineral.unique()]

    for i, mineral in enumerate(minerals):
        codes = minerals_df.loc[minerals_df.mineral == mineral, 'id'].values
        queries = pd.DataFrame(index = [x for x in reporter_df.reporterCode.unique(
        cmdCodes = ','.join([str(x) for x in codes])
        query = ct.getFinalData(
            key, cmdCode=cmdCodes,
            typeCode='C', freqCode='A', clCode='HS', period='2023',
            reporterCode=None,  flowCode='X', partnerCode=0,
            partner2Code=0, customsCode=None, motCode=None
        )
        if query is None or (isinstance(query, pd.DataFrame) and query.empty):
            continue
        query = query.groupby('reporterCode').agg({'qty': 'sum'})
        query.columns = [f'{mineral}']
        trade_df = pd.merge(trade_df, query, left_index=True, right_index=True, how

        if i % 5 == 0:
            print(f'{i+1} of {len(minerals)} complete')

    new_idx = []
    for row_id in trade_df.index:
        new_idx.append(reporter_df.loc[reporter_df.reporterCode == row_id, 'text'].
    trade_df.index = new_idx

    trade_df.to_csv('data/mineral_trade_df.csv')

else:
    trade_df = pd.read_csv('data/mineral_trade_df.csv', index_col=0)

trade_df
```

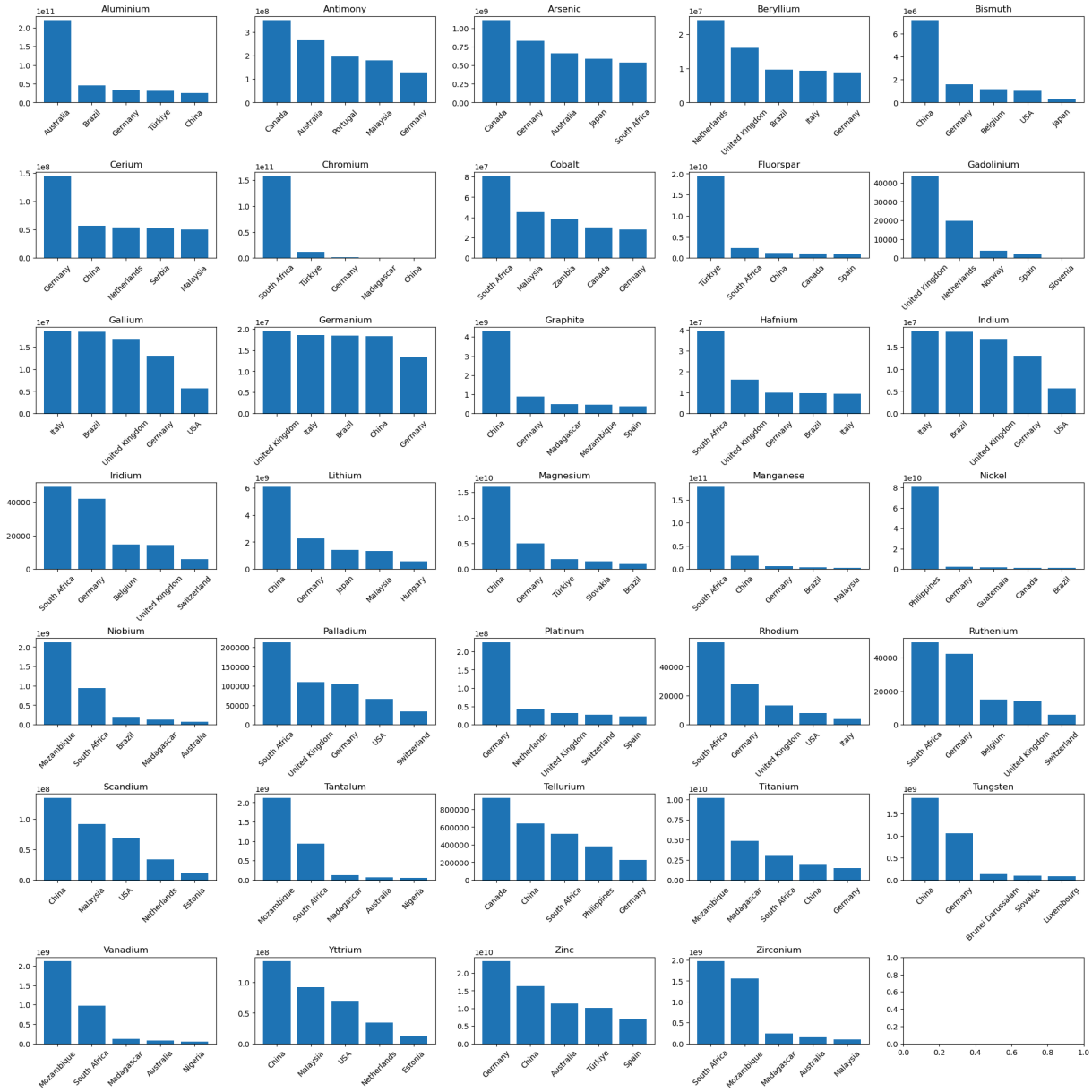
Out[ ]:

	Aluminium	Antimony	Arsenic	Beryllium	Bismuth	Cerium	Chromium
Afghanistan	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Albania	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Algeria	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Andorra	6.559884e+05	0.0	47210.0	0.0	0.0	0.0	0.0
Angola	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
Yemen	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Yugoslavia (...1991)	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Serbia and Montenegro (...2005)	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Zambia	2.885031e+07	26257952.0	104380.0	0.0	0.0	24002.0	0.0
ASEAN	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0

255 rows × 34 columns

In [ ]:

```
fig, axes = plt.subplots(7, 5, figsize = (20, 20), layout = 'constrained')

for col, ax in zip(trade_df.columns, axes.ravel()):
    top5 = trade_df.sort_values(by = col, ascending = False).head(5)
    ax.bar(top5.index, top5[col])
    ax.set_title(col)
    ax.tick_params(axis='x', labelrotation=45)

plt.show()
```



In [ ]:
```
trade_df.sum(axis=1)
```

Out[ ]:
```
Afghanistan                          0.000000e+00
Albania                              0.000000e+00
Algeria                              0.000000e+00
Andorra                              7.689468e+05
Angola                               0.000000e+00
                                         ...
Yemen                                0.000000e+00
Yugoslavia (...1991)                 0.000000e+00
Serbia and Montenegro (...2005)      0.000000e+00
Zambia                               1.919147e+09
ASEAN                                0.000000e+00
Length: 255, dtype: float64
```

Country Data

https://www.census.gov/foreign-trade/data/index.html

In [ ]:
```
url = 'https://api.census.gov/data/timeseries/intltrade/exports/hs'
params = {
    'get': 'ALL_VAL_YR,CTY_CODE,CTY_NAME',
    'YEAR': '2023',
    'MONTH': '12'
}
response = requests.get(url, params=params)
data = response.json()
census_df = pd.DataFrame(data[1:], columns = data[0])
census_df.head()
```

Out[ ]:

	ALL_VAL_YR	CTY_CODE	CTY_NAME	YEAR	MONTH
0	2019159665204	-	TOTAL FOR ALL COUNTRIES	2023	12
1	368755613397	0003	EUROPEAN UNION	2023	12
2	476306245354	0014	PACIFIC RIM COUNTRIES	2023	12
3	45379318929	0017	CAFTA-DR	2023	12
4	676070718257	0020	NAFTA	2023	12

In [ ]:
```
# Standardizing country names by removing punctuation and converting to lowercase f
census_df['CTY_NAME_CLEAN'] = census_df['CTY_NAME'].str.replace('[^\w\s]', '', rege
reporter_df['text_CLEAN'] = reporter_df['text'].str.replace('[^\w\s]', '', regex=Tr

# Merging the datasets based on the cleaned country names
country_mapping = pd.merge(census_df, reporter_df, left_on='CTY_NAME_CLEAN', right_

# Drop unnecessary columns and rename the relevant ones for clarity
country_mapping = country_mapping[['CTY_NAME', 'CTY_CODE', 'text', 'reporterCode']]
country_mapping.rename(columns={
    'CTY_NAME': 'Census Country',
    'CTY_CODE': 'Census Code',
    'text': 'Comtrade Country',
    'reporterCode': 'Comtrade Code'
}, inplace=True)

country_mapping.dropna(how='any')
```

Out[ ]:

	Census Country	Census Code	Comtrade Country	Comtrade Code
1	EUROPEAN UNION	0003	European Union	97.0
11	ASEAN	0027	ASEAN	975.0
13	GREENLAND	1010	Greenland	304.0
14	CANADA	1220	Canada	124.0
17	MEXICO	2010	Mexico	484.0
...	...	...	...	...
247	ZAMBIA	7940	Zambia	894.0
248	ESWATINI	7950	Eswatini	748.0
249	ZIMBABWE	7960	Zimbabwe	716.0
250	MALAWI	7970	Malawi	454.0
251	LESOTHO	7990	Lesotho	426.0

177 rows × 4 columns

In [ ]:

```
unmatched = [cty for cty in reporter_df['text'] if cty not in country_mapping['Comt

matches = {}
for comtrade_cty in unmatched:
    census_match = process.extract(comtrade_cty, census_df['CTY_NAME'].tolist(), li
    if census_match[0][1] > 70:
        matches.update({comtrade_cty: census_match[0][0]})

matches
```

Out[ ]:
```
{'Bosnia Herzegovina': 'BOSNIA AND HERZEGOVINA',
 'Br. Virgin Isds': 'BRITISH VIRGIN ISLANDS',
 'Cayman Isds': 'CAYMAN ISLANDS',
 'Central African Rep.': 'CENTRAL AFRICAN REPUBLIC',
 'Cook Isds': 'COOK ISLANDS',
 "Côte d'Ivoire": "COTE D'IVOIRE",
 'Curaçao': 'CURACAO',
 'Dominican Rep.': 'DOMINICAN REPUBLIC',
 'Faeroe Isds': 'FAROE ISLANDS',
 'FS Micronesia': 'MICRONESIA',
 'Marshall Isds': 'MARSHALL ISLANDS',
 'North Macedonia': 'MACEDONIA',
 'Saint Helena': 'ST HELENA',
 'Saint Kitts and Nevis': 'ST KITTS AND NEVIS',
 'Saint Lucia': 'ST LUCIA',
 'Saint Maarten': 'SINT MAARTEN',
 'Saint Pierre and Miquelon': 'ST PIERRE AND MIQUELON',
 'Saint Vincent and the Grenadines': 'ST VINCENT AND THE GRENADINES',
 'Solomon Isds': 'SOLOMON ISLANDS',
 'Turks and Caicos Isds': 'TURKS AND CAICOS ISLANDS',
 'Viet Nam': 'VIETNAM',
 'Wallis and Futuna Isds': 'WALLIS AND FUTUNA'}
```

In [ ]:
```
for comtrade_cty, census_cty in matches.items():
    census_code = census_df.loc[census_df.CTY_NAME == census_cty, 'CTY_CODE'].value
    comtrade_code = reporter_df.loc[reporter_df.text == comtrade_cty, 'reporterCode
    row = {
        'Census Country': census_cty,
        'Census Code': census_code,
        'Comtrade Country': comtrade_cty,
        'Comtrade Code': comtrade_code
    }
    country_mapping = pd.concat([country_mapping, pd.DataFrame(row, index=[0])], ax

country_mapping.dropna(how='any')
```

Out[ ]:

	Census Country	Census Code	Comtrade Country	Comtrade Code
1	EUROPEAN UNION	0003	European Union	97.0
11	ASEAN	0027	ASEAN	975.0
13	GREENLAND	1010	Greenland	304.0
14	CANADA	1220	Canada	124.0
17	MEXICO	2010	Mexico	484.0
...	...	...	...	...
270	ST VINCENT AND THE GRENADINES	2488	Saint Vincent and the Grenadines	670.0
271	SOLOMON ISLANDS	6223	Solomon Isds	90.0
272	TURKS AND CAICOS ISLANDS	2430	Turks and Caicos Isds	796.0
273	VIETNAM	5520	Viet Nam	704.0
274	WALLIS AND FUTUNA	6413	Wallis and Futuna Isds	876.0

199 rows × 4 columns

In [ ]:
```
trade_df_merged = pd.merge(
    pd.merge(trade_df, country_mapping, left_index=True, right_on='Comtrade Country
    census_df[['CTY_CODE', 'ALL_VAL_YR']], left_on='Census Code', right_on='CTY_COD
)

trade_df_merged.index = trade_df_merged['Comtrade Country']
trade_df_merged = trade_df_merged.drop(columns=['Census Country', 'Census Code', 'C
trade_df_merged = trade_df_merged.astype(float)

trade_df_merged
```

Out[ ]:

Comtrade Country	Aluminium	Antimony	Arsenic	Beryllium	Bismuth	Cerium	Chromium
Afghanistan	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Albania	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Algeria	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Andorra	6.559884e+05	0.0	47210.0	0.0	0.0	0.0	0.0
Angola	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
Yemen	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Yugoslavia (...1991)	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Serbia and Montenegro (...2005)	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Zambia	2.885031e+07	26257952.0	104380.0	0.0	0.0	24002.0	0.0
ASEAN	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0

255 rows × 35 columns

In [ ]:

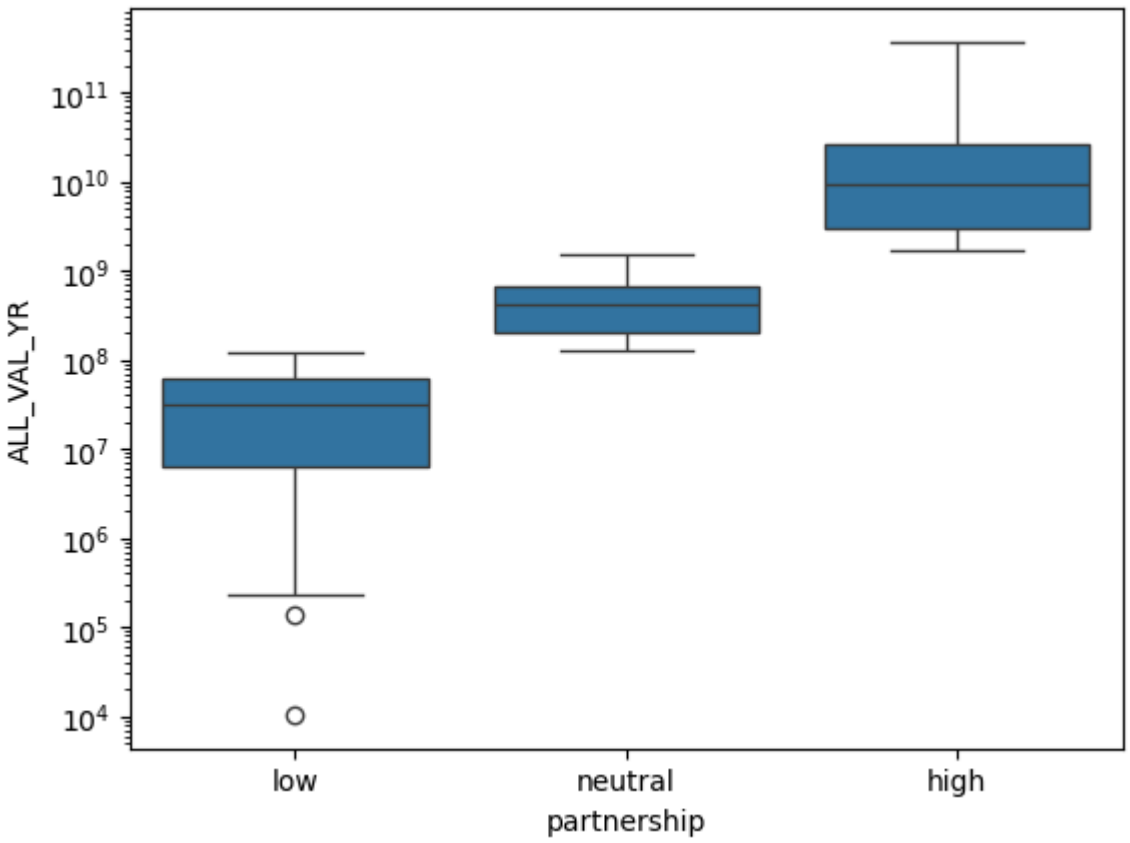
```
quartiles = np.quantile(trade_df_merged.ALL_VAL_YR.dropna(), [0.33, 0.66])

trade_df_merged['partnership'] = trade_df_merged.ALL_VAL_YR.apply(lambda x:
    'na' if np.isnan(x) else
    'low' if x < quartiles[0] else
    # 'neutral_low' if x < quartiles[1] else
    # 'neutral_high' if x < quartiles[2] else
    'neutral' if x < quartiles[1] else
    'high'
)

trade_df_merged
```

Out[ ]:

Comtrade Country	Aluminium	Antimony	Arsenic	Beryllium	Bismuth	Cerium	Chromium
Afghanistan	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Albania	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Algeria	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Andorra	6.559884e+05	0.0	47210.0	0.0	0.0	0.0	0.0
Angola	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
Yemen	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Yugoslavia (...1991)	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Serbia and Montenegro (...2005)	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0
Zambia	2.885031e+07	26257952.0	104380.0	0.0	0.0	24002.0	0.0
ASEAN	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0

255 rows × 36 columns

In [ ]:
```
plt.figure()
sns.boxplot(data=trade_df_merged, x='partnership', y='ALL_VAL_YR', order=['low', 'n
plt.show()
```



In [ ]:
```
trade_df_merged.to_csv('data/mineral_trade_df_full.csv')
```

Alternative characterization of countries

The above characterization, based on trade volume, ultimately proved difficult to work with. It also placed countries like China in the top bin, akin to friendly countries, which clearly misses the point. So, I had to work with some LLMs and manually cobble together a categorization of countries. See the list here:
`https://github.com/kac624/cuny/blob/main/D608/utils.py`

My primary source was this wiki article:
`https://en.wikipedia.org/wiki/Foreign_relations_of_the_United_States#Countrie`

In [ ]:
```
country_relationships = pd.DataFrame(country_relationships, index=[0]).T
country_relationships.columns = ['partnership']
country_relationships.partnership = country_relationships.partnership.apply(lambda

trade_df_merged = pd.merge(
    trade_df, country_relationships,
    left_index=True, right_index=True, how='left'
).dropna(how='any')

print(f'Shape: {trade_df_merged.shape}')
trade_df_merged.head()
```

Shape: (83, 35)

Out[ ]:

	Aluminium	Antimony	Arsenic	Beryllium	Bismuth	Cerium	Chr
Andorra	6.559884e+05	0.000000e+00	4.721000e+04	0.000	0.0	0.00	
Antigua and Barbuda	1.666357e+05	0.000000e+00	0.000000e+00	0.000	0.0	6.28	
Azerbaijan	5.960192e+08	2.490764e+07	5.100435e+05	0.000	0.0	0.00	8
Argentina	3.685202e+07	0.000000e+00	0.000000e+00	0.000	0.0	0.00	28
Australia	2.195847e+11	2.640769e+08	6.610013e+08	12.522	1.2	317661.84	649

5 rows × 35 columns

In [ ]:
```
trade_df_merged.to_csv('data/mineral_trade_df_full.csv')
```

Visualizations

In [ ]:
```
df = trade_df_merged.copy()
df['country'] = df.index

top_minerals = df.drop(['partnership', 'country'], axis=1).sum(axis=0).sort_values(
df = df.drop(
    [x for x in df.columns if x not in top_minerals and
     x != 'partnership' and
     x != 'country'
    ], axis=1
).dropna(how='any')

df.shape
```

Out[ ]: (83, 14)

In [ ]:
```
top10_friendly = (trade_df_merged
    .query('partnership == "Friendly"')
    .drop(['partnership'], axis=1)
    .T.sum(axis=0)
    .sort_values(ascending=False)
    .head(10)
    .index.to_list()
)

top10_neutral = (trade_df_merged
    .query('partnership == "Neutral"')
    .drop(['partnership'], axis=1)
    .T.sum(axis=0)
    .sort_values(ascending=False)
    .head(10)
    .index.to_list()
```

```
)

all_adversarial = trade_df_merged.query('partnership == "Adversarial"').index.to_li

df_sankey = df.loc[(
    df.country.isin(top10_friendly) |
    df.country.isin(top10_neutral) |
    df.country.isin(all_adversarial)
), :]

df_sankey = df_sankey.melt(id_vars=['country', 'partnership'], value_vars=df.column
```

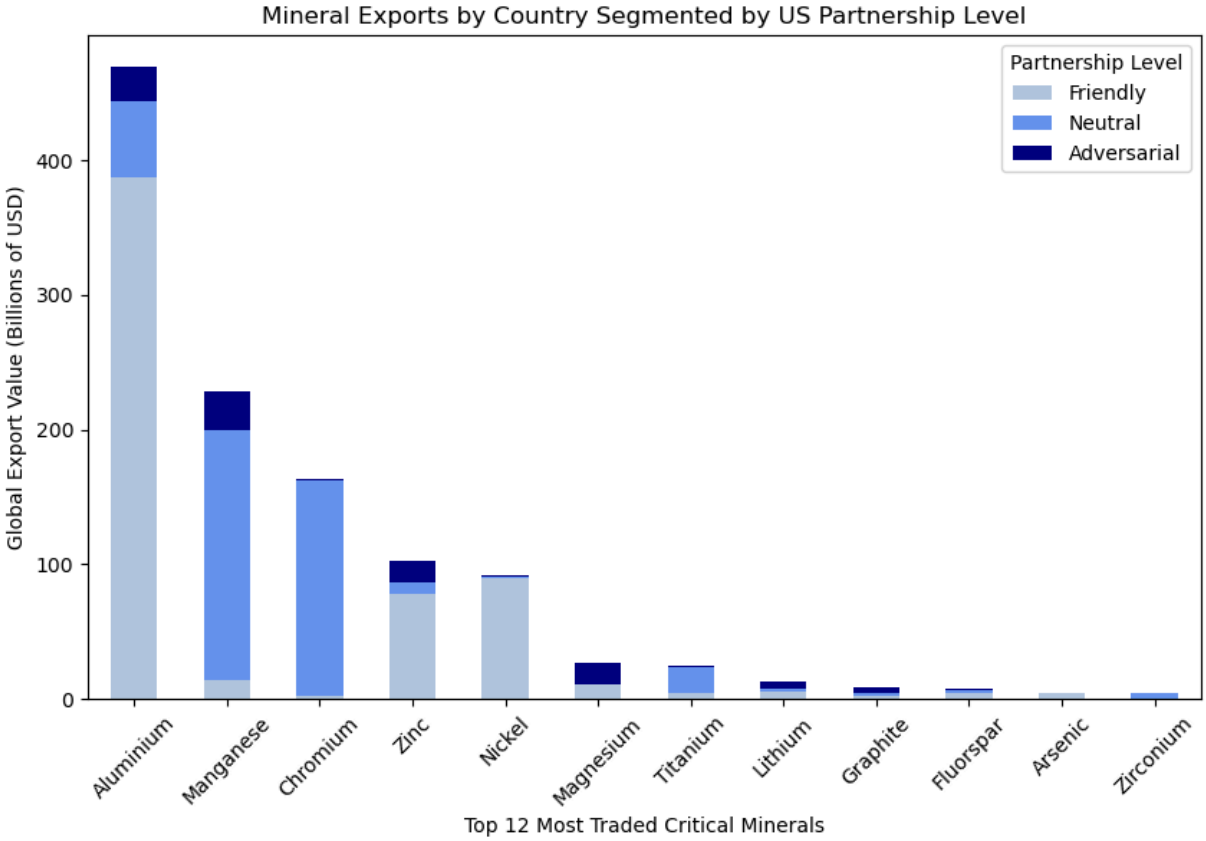
In [ ]:

```
minerals = list(df.columns[:-2])
countries = df['country'].unique()
partnerships = df['partnership'].unique()
labels = minerals + [f"{country} ({partnership})" for country, partnership in zip(d

# Map labels to indices
df_sankey['source'] = df_sankey['mineral'].apply(lambda x: labels.index(x))
df_sankey['target'] = df_sankey.apply(lambda row: labels.index(f"{row['country']} (

# Create a color mapping for each partnership
color_palette = {
    'Friendly': 'lightsteelblue',
    'Neutral': 'cornflowerblue',
    'Adversarial': 'navy'
}
df_sankey['color'] = df_sankey['partnership'].map(color_palette)

# Node positions (manually adjust these based on your data's specific needs)
y_positions = {
    'Friendly': 0.3,
    'Neutral': 0.5,
    'Adversarial': 0.7
}
node_y = [y_positions[partnership] for partnership in df['partnership']]

# Create the Sankey diagram
fig = go.Figure(data=[go.Sankey(
    node=dict(
        pad=20,
        line=dict(color='black', width=0.5),
        label=labels,
        color=[color_palette.get(part, 'blue') for part in df['partnership']],
    ),
    link=dict(
        source=df_sankey['source'].tolist(),
        target=df_sankey['target'].tolist(),
        value=df_sankey['export_value'].tolist(),
        color=df_sankey['color'].tolist()
    ))])

fig.update_layout(
    title_text=f'Mineral Trade Flows to Countries, Segregated by Partnership Level<
    font_size=12,
    height=800,
```

```
    width=1000,
    margin=dict(l=20, r=150, t=70, b=70)
)
fig.show()
```

In [ ]:

```
partnership_colors = {
    'Friendly': 'lightsteelblue',
    'Neutral': 'cornflowerblue',
    'Adversarial': 'navy'
}

# Aggregate export values by partnership and pivot
pivot_df = df.groupby(['partnership']).sum().reset_index().drop(['country'], axis=1
pivot_df = pivot_df.melt(id_vars='partnership')
pivot_df = pivot_df.pivot(index='variable', columns='partnership', values='value')

# Sort by total exports
pivot_df['total_exports'] = pivot_df.sum(axis=1)
pivot_df = pivot_df.sort_values(by='total_exports', ascending=False)
pivot_df = pivot_df.drop('total_exports', axis=1)
pivot_df = pivot_df[['Friendly', 'Neutral', 'Adversarial']]
pivot_df = pivot_df / (10**9)

# Plotting
fig, ax = plt.subplots(figsize=(10, 6))
pivot_df.plot(kind='bar', stacked=True, ax=ax, color=partnership_colors)
plt.title('Mineral Exports by Country Segmented by US Partnership Level')
plt.ylabel('Global Export Value (Billions of USD)')
plt.xlabel('Top 12 Most Traded Critical Minerals')
plt.xticks(rotation=45)
plt.legend(title='Partnership Level')
plt.show()
```


Mineral Exports by Country Segmented by US Partnership Level
Partnership Level
Friendly
Neutral
Adversarial
Global Export Value (Billions of USD)
0
100
200
300
400
Aluminium
Manganese
Chromium
Zinc
Nickel
Magnesium
Titanium
Lithium
Graphite
Fluorspar
Arsenic
Zirconium
Top 12 Most Traded Critical Minerals