Lab Assignment 9: Data Management Using `pandas`, Part 2

DS 6001: Practice and Application of Data Science

Instructions

Please answer the following questions as completely as possible using text, code, and the results of code as needed. Format your answers in a Jupyter notebook. To receive full credit, make sure you address every part of the problem, and make sure your document is formatted in a clean and professional way.

In this lab, we are going to build the Country Analysis Relational DataBase (which we will call the C.A.R.D.B. or the "Cardi B"):

We will be collecting data from two sources. First, we will use open data from the World Bank's Sovereign Environmental, Social, and Governance (ESG) Data project. The ESG data reports data from every country in the world over the time frame from 1960-2022 on a wide variety of topics including education, health, and economic factors within the countries. Second, we will use data on the quality and democratic character of countries' governments as reported by the Varieties of Democracy (V-Dem) project at the University of Notre Dame. By using both data sources, we can conduct analyses to see whether democratic openness leads to better societal outcomes for countries. We can also write queries to capture a wide range of information on countries' political parties, tax systems, and banking industries, for example. Or as Cardi B would say, "You in the club just to party, I'm there, I get paid a fee. I be in and out them banks so much, I know they're tired of me."

Problem 0

Import the following packages (use `pip install` to download any packages you don't already have installed):

In [1]:
```
import numpy as np
import pandas as pd
import requests
import os
import io
import zipfile
```

Both the World Bank and V-Dem store their data in zipped directories containing CSV files. Download the World Bank data into your current working directory by typing the following code:

In [2]:
```
url = 'https://databank.worldbank.org/data/download/ESG_CSV.zip'
r = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(r.content))
z.extractall()
```

And download the V-Dem data by typing:

In [3]:
```
url = 'https://v-dem.net/media/datasets/V-Dem-CY-Core_csv_v13.zip'
r = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(r.content))
z.extractall()
```

After you've run this code successfully once, the files you need will be in your working directory and you should save time by switching these cells from "code" to "raw" so that they don't run again if you restart the kernel.

You will only need three of the files you've downloaded. Load the 'V-Dem-CY-Core-v13.csv' file as `vdem` and the 'ESGData.csv' file as `wb` .

In [4]:
```
vdem = pd.read_csv('V-Dem-CY-Core-v13.csv')
wb = pd.read_csv('ESGCSV.csv')
```

Problem 1

First, let's focus on the `vdem` data ('V-Dem-CY-Core-v13.csv'). Use `pandas` methods to perform the following tasks:

Part a

Keep only the 'country_text_id', 'country_name','year', 'v2x_polyarchy', and 'v2peedueq' columns. [1 point]

In [5]:
```
vdem = vdem[['country_text_id', 'country_name', 'year', 'v2x_polyarchy', 'v2
vdem
```

Out[5]:

	country_text_id	country_name	year	v2x_polyarchy	v2peedueq
0	MEX	Mexico	1789	0.028	NaN
1	MEX	Mexico	1790	0.028	NaN
2	MEX	Mexico	1791	0.028	NaN
3	MEX	Mexico	1792	0.028	NaN
4	MEX	Mexico	1793	0.028	NaN
...	...	...	...	...	...
27550	SPD	Piedmont-Sardinia	1857	0.207	NaN
27551	SPD	Piedmont-Sardinia	1858	0.210	NaN
27552	SPD	Piedmont-Sardinia	1859	0.210	NaN
27553	SPD	Piedmont-Sardinia	1860	0.213	NaN
27554	SPD	Piedmont-Sardinia	1861	0.213	NaN

27555 rows × 5 columns

Part b

Use the `.query()` method to keep only the rows in which year is greater than or equal to 1960 and less than or equal to 2021. [1 point]

In [6]:
```
vdem = vdem.query("year >= 1960 & year <= 2021")
vdem
```

Out [6]:

	country_text_id	country_name	year	v2x_polyarchy	v2peedueq
171	MEX	Mexico	1960	0.232	-1.438
172	MEX	Mexico	1961	0.234	-1.438
173	MEX	Mexico	1962	0.233	-1.438
174	MEX	Mexico	1963	0.233	-1.438
175	MEX	Mexico	1964	0.231	-1.438
...	...	...	...	...	...
26150	ZZB	Zanzibar	2017	0.267	1.661
26151	ZZB	Zanzibar	2018	0.268	1.486
26152	ZZB	Zanzibar	2019	0.266	1.486
26153	ZZB	Zanzibar	2020	0.258	1.427
26154	ZZB	Zanzibar	2021	0.276	1.779

10371 rows × 5 columns

Part c

Rename 'country_text_id' to 'country_code', 'country_name' to 'country_name_vdem', 'v2x_polyarchy' to 'democracy', and 'v2peedueq' to 'educational_equality'. [1 point]

In [7]:

```
vdem = vdem.rename({'country_text_id': "country_code",
                    'country_name': "country_name_vdem",
                    'v2x_polyarchy': "democracy",
                    'v2peedueq': "educational_equality"}, axis = 1)
vdem
```

Out [7]:

	country_code	country_name_vdem	year	democracy	educational_equality
171	MEX	Mexico	1960	0.232	-1.438
172	MEX	Mexico	1961	0.234	-1.438
173	MEX	Mexico	1962	0.233	-1.438
174	MEX	Mexico	1963	0.233	-1.438
175	MEX	Mexico	1964	0.231	-1.438
...	...	...	...	...	...
26150	ZZB	Zanzibar	2017	0.267	1.661
26151	ZZB	Zanzibar	2018	0.268	1.486
26152	ZZB	Zanzibar	2019	0.266	1.486
26153	ZZB	Zanzibar	2020	0.258	1.427
26154	ZZB	Zanzibar	2021	0.276	1.779

10371 rows × 5 columns

Part d

Sort the rows by 'country_code' and 'year' in ascending order. [1 point]

In [8]:

```
vdem = vdem.sort_values(['country_code', 'year'], ascending=[True, True]).re
vdem
```

Out [8]:

	country_code	country_name_vdem	year	democracy	educational_equality
0	AFG	Afghanistan	1960	0.080	-1.123
1	AFG	Afghanistan	1961	0.083	-1.123
2	AFG	Afghanistan	1962	0.082	-1.123
3	AFG	Afghanistan	1963	0.085	-1.123
4	AFG	Afghanistan	1964	0.137	-0.951
...	...	...	...	...	...
10366	ZZB	Zanzibar	2017	0.267	1.661
10367	ZZB	Zanzibar	2018	0.268	1.486
10368	ZZB	Zanzibar	2019	0.266	1.486
10369	ZZB	Zanzibar	2020	0.258	1.427
10370	ZZB	Zanzibar	2021	0.276	1.779

10371 rows × 5 columns

Problem 2

Next focus on the World Bank `wb` dataset 'ESGData.csv'. Use `pandas` methods to perform the following tasks:

Part a

Keep only the columns named 'Country Code', 'Country Name', and 'Indicator Code', or begin with '19' or '20'. (Don't type in all the years individually. Instead, use code that finds all columns that begin '19' or '20'.) [1 point]

In [9]:
```
wb_cols = [x for x in wb.columns if x.startswith("19") or x.startswith("20")
cols = ['Country Code', 'Country Name', 'Indicator Code']
wb = wb[cols+wb_cols]
wb
```

Out[9]:

	Country Code	Country Name	Indicator Code	1960	1961	1962	196
0	ARB	Arab World	EG.CFT.ACCS.ZS	NaN	NaN	NaN	Na
1	ARB	Arab World	EG.ELC.ACCS.ZS	NaN	NaN	NaN	Na
2	ARB	Arab World	NY.ADJ.DRES.GN.ZS	NaN	NaN	NaN	Na
3	ARB	Arab World	NY.ADJ.DFOR.GN.ZS	NaN	NaN	NaN	Na
4	ARB	Arab World	AG.LND.AGRI.ZS	NaN	30.981414	30.982663	31.00705
...	...	...	...	...	...	...	
16964	ZWE	Zimbabwe	ER.PTD.TOTL.ZS	NaN	NaN	NaN	Na
16965	ZWE	Zimbabwe	AG.LND.FRLS.HA	NaN	NaN	NaN	Na
16966	ZWE	Zimbabwe	SL.UEM.TOTL.ZS	NaN	NaN	NaN	Na
16967	ZWE	Zimbabwe	SP.UWT.TFRT	NaN	NaN	NaN	Na
16968	ZWE	Zimbabwe	VA.EST	NaN	NaN	NaN	Na

16969 rows × 67 columns

Part b

Rename 'Country Code' to 'country_code', 'Country Name' to 'country_name_wb', and 'Indicator Code' to 'feature'. [1 point]

In [10]:
```
wb = wb.rename({'Country Code': "country_code",
                'Country Name': "country_name_wb",
                'Indicator Code': 'feature'}, axis = 1)
wb
```

Out[10]:

	country_code	country_name_wb	feature	1960	1961	1
0	ARB	Arab World	EG.CFT.ACCS.ZS	NaN	NaN	
1	ARB	Arab World	EG.ELC.ACCS.ZS	NaN	NaN	
2	ARB	Arab World	NY.ADJ.DRES.GN.ZS	NaN	NaN	
3	ARB	Arab World	NY.ADJ.DFOR.GN.ZS	NaN	NaN	
4	ARB	Arab World	AG.LND.AGRI.ZS	NaN	30.981414	30.982
...	...	...	...	...	...	
16964	ZWE	Zimbabwe	ER.PTD.TOTL.ZS	NaN	NaN	
16965	ZWE	Zimbabwe	AG.LND.FRLS.HA	NaN	NaN	
16966	ZWE	Zimbabwe	SL.UEM.TOTL.ZS	NaN	NaN	
16967	ZWE	Zimbabwe	SP.UWT.TFRT	NaN	NaN	
16968	ZWE	Zimbabwe	VA.EST	NaN	NaN	

16969 rows × 67 columns

Part c

Use the `.query()` method to remove the rows in which 'country_name_wb' is equal to one of the entries in the folowing `noncountries` list: [1 point]

In [11]:
```
noncountries = ["Arab World", "Central Europe and the Baltics",
                "Caribbean small states",
                "East Asia & Pacific (excluding high income)",
                "Early-demographic dividend","East Asia & Pacific",
                "Europe & Central Asia (excluding high income)",
                "Europe & Central Asia", "Euro area",
                "European Union","Fragile and conflict affected situations",
                "High income",
                "Heavily indebted poor countries (HIPC)","IBRD only",
                "IDA & IBRD total",
                "IDA total","IDA blend","IDA only",
                "Latin America & Caribbean (excluding high income)",
                "Latin America & Caribbean",
                "Least developed countries: UN classification",
                "Low income","Lower middle income","Low & middle income",
                "Late-demographic dividend","Middle East & North Africa",
                "Middle income",
                "Middle East & North Africa (excluding high income)",
                "North America","OECD members",
                "Other small states","Pre-demographic dividend",
```

```
                 "Pacific island small states",
                 "Post-demographic dividend",
                 "Sub-Saharan Africa (excluding high income)",
                 "Sub-Saharan Africa",
                 "Small states","East Asia & Pacific (IDA & IBRD)",
                 "Europe & Central Asia (IDA & IBRD)",
                 "Latin America & Caribbean (IDA & IBRD)",
                 "Middle East & North Africa (IDA & IBRD)","South Asia",
                 "South Asia (IDA & IBRD)",
                 "Sub-Saharan Africa (IDA & IBRD)",
                 "Upper middle income", "World"]
```

In [12]:

```
wb = wb.query("country_name_wb not in @noncountries")
wb
```

Out[12]:

	country_code	country_name_wb	feature	1960	1961	1!
3266	AFG	Afghanistan	EG.CFT.ACCS.ZS	NaN	NaN	I
3267	AFG	Afghanistan	EG.ELC.ACCS.ZS	NaN	NaN	I
3268	AFG	Afghanistan	NY.ADJ.DRES.GN.ZS	NaN	NaN	I
3269	AFG	Afghanistan	NY.ADJ.DFOR.GN.ZS	NaN	NaN	I
3270	AFG	Afghanistan	AG.LND.AGRI.ZS	NaN	57.878356	57.955
...	...	...	...	...	...	
16964	ZWE	Zimbabwe	ER.PTD.TOTL.ZS	NaN	NaN	I
16965	ZWE	Zimbabwe	AG.LND.FRLS.HA	NaN	NaN	I
16966	ZWE	Zimbabwe	SL.UEM.TOTL.ZS	NaN	NaN	I
16967	ZWE	Zimbabwe	SP.UWT.TFRT	NaN	NaN	I
16968	ZWE	Zimbabwe	VA.EST	NaN	NaN	I

13703 rows × 67 columns

Part d

The features in this dataset are given strange and incomprehensible codes such as 'EG.CFT.ACCS.ZS'. Use the `replace_map` dictionary, defined below, to recode all of these values with more descriptive names for each feature. [1 point]

In [13]:

```
replace_map = {
  "AG.LND.AGRI.ZS": "agricultural_land",
  "AG.LND.FRST.ZS": "forest_area",
  "AG.PRD.FOOD.XD": "food_production_index",
  "CC.EST": "control_of_corruption",
  "EG.CFT.ACCS.ZS": "access_to_clean_fuels_and_technologies_for_cooking",
  "EG.EGY.PRIM.PP.KD": "energy_intensity_level_of_primary_energy",
  "EG.ELC.ACCS.ZS": "access_to_electricity",
  "EG.ELC.COAL.ZS": "electricity_production_from_coal_sources",
```

```
    "EG.ELC.RNEW.ZS": "renewable_electricity_output",
    "EG.FEC.RNEW.ZS": "renewable_energy_consumption",
    "EG.IMP.CONS.ZS": "energy_imports",
    "EG.USE.COMM.FO.ZS": "fossil_fuel_energy_consumption",
    "EG.USE.PCAP.KG.OE": "energy_use",
    "EN.ATM.CO2E.PC": "co2_emissions",
    "EN.ATM.METH.PC": "methane_emissions",
    "EN.ATM.NOXE.PC": "nitrous_oxide_emissions",
    "EN.ATM.PM25.MC.M3": "pm2_5_air_pollution",
    "EN.CLC.CDDY.XD": "cooling_degree_days",
    "EN.CLC.GHGR.MT.CE": "ghg_net_emissions",
    "EN.CLC.HEAT.XD": "heat_index_35",
    "EN.CLC.MDAT.ZS": "droughts",
    "EN.CLC.PRCP.XD": "maximum_5-day_rainfall",
    "EN.CLC.SPEI.XD": "mean_drought_index",
    "EN.MAM.THRD.NO": "mammal_species",
    "EN.POP.DNST": "population_density",
    "ER.H2O.FWTL.ZS": "annual_freshwater_withdrawals",
    "ER.PTD.TOTL.ZS": "terrestrial_and_marine_protected_areas",
    "GB.XPD.RSDV.GD.ZS": "research_and_development_expenditure",
    "GE.EST": "government_effectiveness",
    "IC.BUS.EASE.XQ": "ease_of_doing_business_rank",
    "IC.LGL.CRED.XQ": "strength_of_legal_rights_index",
    "IP.JRN.ARTC.SC": "scientific_and_technical_journal_articles",
    "IP.PAT.RESD": "patent_applications",
    "IT.NET.USER.ZS": "individuals_using_the_internet",
    "NV.AGR.TOTL.ZS": "agriculture",
    "NY.ADJ.DFOR.GN.ZS": "net_forest_depletion",
    "NY.ADJ.DRES.GN.ZS": "natural_resources_depletion",
    "NY.GDP.MKTP.KD.ZG": "gdp_growth",
    "PV.EST": "political_stability_and_absence_of_violence",
    "RL.EST": "rule_of_law",
    "RQ.EST": "regulatory_quality",
    "SE.ADT.LITR.ZS": "literacy_rate",
    "SE.ENR.PRSC.FM.ZS": "gross_school_enrollment",
    "SE.PRM.ENRR": "primary_school_enrollment",
    "SE.XPD.TOTL.GB.ZS": "government_expenditure_on_education",
    "SG.GEN.PARL.ZS": "proportion_of_seats_held_by_women_in_national_parliamen
    "SH.DTH.COMM.ZS": "cause_of_death",
    "SH.DYN.MORT": "mortality_rate",
    "SH.H2O.SMDW.ZS": "people_using_safely_managed_drinking_water_services",
    "SH.MED.BEDS.ZS": "hospital_beds",
    "SH.STA.OWAD.ZS": "prevalence_of_overweight",
    "SH.STA.SMSS.ZS": "people_using_safely_managed_sanitation_services",
    "SI.DST.FRST.20": "income_share_held_by_lowest_20pct",
    "SI.POV.GINI": "gini_index",
    "SI.POV.NAHC": "poverty_headcount_ratio_at_national_poverty_lines",
    "SI.SPR.PCAP.ZG": "annualized_average_growth_rate_in_per_capita_real_surve
    "SL.TLF.0714.ZS": "children_in_employment",
    "SL.TLF.ACTI.ZS": "labor_force_participation_rate",
    "SL.TLF.CACT.FM.ZS": "ratio_of_female_to_male_labor_force_participation_ra
    "SL.UEM.TOTL.ZS": "unemployment",
    "SM.POP.NETM": "net_migration",
    "SN.ITK.DEFC.ZS": "prevalence_of_undernourishment",
    "SP.DYN.LE00.IN": "life_expectancy_at_birth",
    "SP.DYN.TFRT.IN": "fertility_rate",
```

```
    "SP.POP.65UP.TO.ZS": "population_ages_65_and_above",
    "SP.UWT.TFRT": "unmet_need_for_contraception",
    "VA.EST": "voice_and_accountability",
    "EN.CLC.CSTP.ZS": "coastal_protection",
    "SD.ESR.PERF.XQ": "economic_and_social_rights_performance_score",
    "EN.CLC.HDDY.XD": "heating_degree_days",
    "EN.LND.LTMP.DC": "land_surface_temperature",
    "ER.H2O.FWST.ZS": "freshwater_withdrawal",
    "EN.H2O.BDYS.ZS": "water_quality",
    "AG.LND.FRLS.HA": "tree_cover_loss",
}
```

In [14]:
```
wb = wb.replace(replace_map)
wb
```

Out[14]:

	country_code	country_name_wb	featu
3266	AFG	Afghanistan	access_to_clean_fuels_and_technologies_for_coo
3267	AFG	Afghanistan	access_to_electrici
3268	AFG	Afghanistan	natural_resources_depleti
3269	AFG	Afghanistan	net_forest_depleti
3270	AFG	Afghanistan	agricultural_la
...	...	...	
16964	ZWE	Zimbabwe	terrestrial_and_marine_protected_are
16965	ZWE	Zimbabwe	tree_cover_lo
16966	ZWE	Zimbabwe	unemployme
16967	ZWE	Zimbabwe	unmet_need_for_contracepti
16968	ZWE	Zimbabwe	voice_and_accountabili

13703 rows × 67 columns

Problem 3

The wb dataset is strangely organized. The features are stored in the rows, when typically we would want these features to be columns. Also, years are stored in columns, when typically we would want years to be represented by different rows. We can repair this structure by reshaping the data.

Part a

First, reshape the data to turn the columns that refer to years into rows. [1 point]

In [15]:
```
wb = pd.melt(wb, ['country_code', 'country_name_wb', 'feature'], [str(i) for
wb
```

Out[15]:

	country_code	country_name_wb	feat
0	AFG	Afghanistan	access_to_clean_fuels_and_technologies_for_cc
1	AFG	Afghanistan	access_to_electri
2	AFG	Afghanistan	natural_resources_deple
3	AFG	Afghanistan	net_forest_deple
4	AFG	Afghanistan	agricultural_l
...	...	...	
863284	ZWE	Zimbabwe	terrestrial_and_marine_protected_ar
863285	ZWE	Zimbabwe	tree_cover_
863286	ZWE	Zimbabwe	unemploym
863287	ZWE	Zimbabwe	unmet_need_for_contracep
863288	ZWE	Zimbabwe	voice_and_accountab

863289 rows × 5 columns

Part b

Then rename `variable` to `year` , and reshape the data again by turning the rows that refer to features into columns. [1 point]

In [16]:

```
wb = wb.rename({'variable': "year"}, axis = 1)
wb = wb.pivot_table(index=['country_code', 'country_name_wb', 'year'], colum
wb = pd.DataFrame(wb.to_records())
wb
```

Out[16]:

	country_code	country_name_wb	year	access_to_clean_fuels_and_technologie
0	AFG	Afghanistan	1960	
1	AFG	Afghanistan	1961	
2	AFG	Afghanistan	1962	
3	AFG	Afghanistan	1963	
4	AFG	Afghanistan	1964	
...	...	...	...	
12154	ZWE	Zimbabwe	2018	
12155	ZWE	Zimbabwe	2019	
12156	ZWE	Zimbabwe	2020	
12157	ZWE	Zimbabwe	2021	
12158	ZWE	Zimbabwe	2022	

12159 rows × 74 columns

Part c

After these reshapes, the year column in the `wb` data frame is stored as a string. Convert this column to an integer data type. [1 point]

In [17]:

```
wb['year'] = wb['year'].astype('int64')
wb.dtypes
```

Out[17]:

```
country_code                                          object
country_name_wb                                       object
year                                                   int64
access_to_clean_fuels_and_technologies_for_cooking   float64
access_to_electricity                                float64
                                                      ...
tree_cover_loss                                      float64
unemployment                                         float64
unmet_need_for_contraception                         float64
voice_and_accountability                             float64
water_quality                                        float64
Length: 74, dtype: object
```

Problem 4

Next we will merge the `wb` data frame with the `vdem` data frame, matching on the 'country_code' and 'year' columns.

Part a

First, write a sentence stating whether you expect this merge to be one-to-one, many-to-one, one-to-many, or many-to-many, and describe your rationale. [1 point]

The merge should be one-to-one, since country code and year serves as a sort of primary key for vdem and wb.

Part b

Next, merge the two datasets together in a way that checks whether your expectation is met, and also allows you to see the rows that failed to match. [2 points]

In [21]:
```
wbvdem = pd.merge(wb, vdem,
         on = ['country_code','year'],
         how = 'outer',
         validate = 'one_to_one',
         indicator = 'matched')
```

Part c

After this merge, use the `.value_counts()` method to see the total number of observations that were found in both datasets, the number found only in the left dataset, and the number found only in the right dataset. (If you entered the `wb` data frame into the merge function first, then "left_only" refers to the rows found in the World Bank but not V-Dem, and "right_only" refers to the rows found in V-Dem but not the World Bank.) There should be more than 9000 rows that matched, but more than 2000 that failed to match.

Then conduct two data aggregations to help us investigate why these observations did not match:

- First use `.query()` to keep only the observations that were present in `wb` but not `vdem`. (These are the 'left_only' observations if you typed the World Bank data into the merge function first.) Use `.groupby()` to aggregate the data by both 'country_code' and 'country_name_wb'. Then save the minimum and maximum values of 'year' for each country.
- Then use `.query()` to keep only the observations that were present in `vdem` data but not `wb`. Use `.groupby()` to aggregate the data by both 'country_code' and 'country_name_vdem'. Then save the minimum and maximum values of 'year' for each country. [2 points]

In [22]:
```
wbvdem['matched'].value_counts()
```

Out[22]:
```
matched
both          9976
left_only     2183
right_only     395
Name: count, dtype: int64
```

In [48]:
```
wbonly = wbvdem.query("matched == 'left_only'")
wbonly = wbonly.groupby(['country_code','country_name_wb']).year.agg(['min',
wbonly
```

Out[48]:

country_code	country_name_wb	min	max
AFG	Afghanistan	2022	2022
AGO	Angola	2022	2022
ALB	Albania	2022	2022
AND	Andorra	1960	2022
ARE	United Arab Emirates	1960	2022
...	...	...	...
WSM	Samoa	1960	2022
YEM	Yemen, Rep.	2022	2022
ZAF	South Africa	2022	2022
ZMB	Zambia	2022	2022
ZWE	Zimbabwe	2022	2022

193 rows × 2 columns

In [46]:
```
vdemonly = wbvdem.query("matched == 'right_only'")
vdemonly = vdemonly.groupby(['country_code','country_name_vdem']).year.agg([
vdemonly
```

Out[46]:

country_code	country_name_vdem	min	max
DDR	German Democratic Republic	1960	1990
HKG	Hong Kong	1960	2021
PSE	Palestine/West Bank	1967	2021
PSG	Palestine/Gaza	1960	2021
SML	Somaliland	1991	2021
TWN	Taiwan	1960	2021
VDR	Republic of Vietnam	1960	1975
XKX	Kosovo	1999	2021
YMD	South Yemen	1960	1990
ZZB	Zanzibar	1960	2021

Part d

Here's where a deep understanding of the data becomes very important. There are two reasons why an observation may fail to match in a merge. One reason is a difference in spelling. Suppose that South Korea (which is also known as the Republic of Korea) is coded as SKO in the World Bank data and ROK in V-Dem. In this case, we should recode one or the other of SKO and ROK so that they match, otherwise we will lose the data on South Korea. But the second reason why observations might fail to match is due to differences in coverage in the data collection strategy: it is possible that a country wasn't included in one data's coverage, or that certain years for that country were not included. For differences in coverage, there's no way to manipulate the data to match, so we are out of luck and we have to either delete these observations or proceed with missing data from one of the data sources.

Take a close look at the two data aggregation tables you generated in part (j), and answer the following questions:

- Do you see any countries that are present in both the unmatched World Bank rows and the unmatched V-Dem rows, but with different spellings?
- Do some digging on Wikipedia and other sources on the Internet. What do you think is the primary reason why some countries are present in the V-Dem data but not the World Bank? (You don't need to describe the reasoning for every country. Just dig until you see a general pattern and describe it here.)
- Do some more digging on Wikipedia and other sources on the Internet. What do you think is the primary reason why some countries are present in the World Bank data

> but not V-Dem? (You don't need to describe the reasoning for every country. Just dig until you see a general pattern and describe it here.) [1 point]

There are no countries in both the unmatched World Bank rows and the unmatched V-Dem rows with different spellings.

The primary reason countries are present in V-Dem but not World Bank seems to be that those countries either no longer exist as an autonomous state (as with South Yemen) or have their existence as an autonomous state disputed (as with Taiwan).

The primary reason countries are present in World Bank but not V-Dem, other than the data values beyond 2021 which does not exist in the V-Dem data, seems to be that those countries did not exist at some point between 1960 and 2021 (for example countries of the former Soviet Union).

Part e

Once you are convinced that all of the unmatched observations are due to differences in the coverage of the data collection strategies of the World Bank and V-Dem, repeat the merge, dropping all unmatched observations. This time there is no need to validate the type of merge, and no need to define a variable to indicate matching. [1 point]

In [65]:
```
wbvdem = pd.merge(wb, vdem,
         on = ['country_code','year'],
         how = 'inner')
wbvdem
```

Out[65]:

	country_code	country_name_wb	year	access_to_clean_fuels_and_technologies
0	AFG	Afghanistan	1960	
1	AFG	Afghanistan	1961	
2	AFG	Afghanistan	1962	
3	AFG	Afghanistan	1963	
4	AFG	Afghanistan	1964	
...	...	...	...	
9971	ZWE	Zimbabwe	2017	
9972	ZWE	Zimbabwe	2018	
9973	ZWE	Zimbabwe	2019	
9974	ZWE	Zimbabwe	2020	
9975	ZWE	Zimbabwe	2021	

9976 rows × 77 columns

Problem 5

Write code using pandas that answers the next two questions:

Part a

Of all countries in the data, which countries have the highest and lowest average levels of democratic quality across the 1960-2022 timespan? [1 point]

In [129…

```
high = pd.DataFrame(wbvdem.groupby(["country_code"])[
             'democracy'].mean().sort_values(
                 ascending=False)).reset_index()[:1]['country_code'].array[0
print("The country with the highest average level of democratic quality is "
      wbvdem.query(f'country_code == "{high}"')[:1]['country_name_wb'].array

low = pd.DataFrame(wbvdem.groupby(["country_code"])[
             'democracy'].mean().sort_values(
                 )).reset_index()[:1]['country_code'].array[0]
print("The country with the lowest average level of democratic quality is "
      wbvdem.query(f'country_code == "{low}"')[:1]['country_name_wb'].array[
```

```
The country with the highest average level of democratic quality is Denmark
The country with the lowest average level of democratic quality is Saudi Ara
bia
```

Part b

The 'educational_equality' index compiled by V-Dem measures the extent to which "high quality basic education guaranteed to all, sufficient to enable them to exercise their basic rights as adult citizens." They use a Bayesian scaling method to create a score for each country in each year that ranges roughly from -4 to 4, where low values of the scale mean that

> Provision of high quality basic education is extremely unequal and at least 75 percent (%) of children receive such low-quality education that undermines their ability to exercise their basic rights as adult citizens.

And high values mean that

> Basic education is equal in quality and less than five percent (%) of children receive such low-quality education that probably undermines their ability to exercise their basic rights as adult citizens.

Use the `pd.cut()` method to create a categorical version of 'educational_equality' with five categories, one from -4 to -2 called "extremely unequal", one from -2 to -.5 called "very unequal", one from -.5 to .5 called "somewhat unequal", one from .5 to 1.5 called "relatively equal", and one for values from 1.5 to 4 called "equal". (By default, the `pd.cut()` method sets `right=True`, which means the bins include their rightmost edges, so a value of exactly -2 will fall within the "extremely unequal" bin. Leave this default in place.)

Then aggregate the data to have one row per category of the new categorical version of "educational_equality". Collapse the following features to the mean with each category of "educational_equality":

- 'gini_index': The GINI index measures the amount of economic inequality in a country. The higher the index, the greater the economic disparity between rich and poor.
- 'poverty_headcount_ratio_at_national_poverty_lines': a measure of the proportion of the population living in poverty [1 point]

In [141…

```
bins = [-4, -2, -.5, .5, 1.5, 4]
labels = ["extremely unequal", "very unequal", "somewhat unequal", "relative
wbvdem['educational_equality_cat'] = pd.DataFrame(pd.cut(wbvdem['educational
wbvdem.groupby(["educational_equality_cat"])[['gini_index', 'poverty_headcou
```

```
/var/folders/ds/qp3gbx7n3tz0738b8w4wxs580000gn/T/ipykernel_63355/183218413
0.py:4: FutureWarning: The default of observed=False is deprecated and will
be changed to True in a future version of pandas. Pass observed=False to ret
ain current behavior or observed=True to adopt the future default and silenc
e this warning.
  wbvdem.groupby(["educational_equality_cat"])[['gini_index', 'poverty_headc
ount_ratio_at_national_poverty_lines']].mean()
```

Out[141…

educational_equality_cat	gini_index	poverty_headcount_ratio_at_national_poverty_lines
extremely unequal	38.846154	58.16000(
very unequal	45.926484	38.63605ε
somewhat unequal	43.200442	24.14912ɜ
relatively equal	37.148861	22.54853ϵ
equal	32.652901	17.20744ʌ

In [ ]: