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 two 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')
```

In [5]:
```
wb
```

Out[5]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962
0	Arab World	ARB	Access to clean fuels and technologies for coo...	EG.CFT.ACCS.ZS	NaN	NaN	NaN
1	Arab World	ARB	Access to electricity (% of population)	EG.ELC.ACCS.ZS	NaN	NaN	NaN
2	Arab World	ARB	Adjusted savings: natural resources depletion ...	NY.ADJ.DRES.GN.ZS	NaN	NaN	NaN
3	Arab World	ARB	Adjusted savings: net forest depletion (% of GNI)	NY.ADJ.DFOR.GN.ZS	NaN	NaN	NaN
4	Arab World	ARB	Agricultural land (% of land area)	AG.LND.AGRI.ZS	NaN	30.981414	30.982663
...	...	...	...	...	...	...	...
16964	Zimbabwe	ZWE	Terrestrial and marine protected areas (% of t...	ER.PTD.TOTL.ZS	NaN	NaN	NaN
16965	Zimbabwe	ZWE	Tree Cover Loss (hectares)	AG.LND.FRLS.HA	NaN	NaN	NaN
16966	Zimbabwe	ZWE	Unemployment, total (% of total labor force) (...	SL.UEM.TOTL.ZS	NaN	NaN	NaN
16967	Zimbabwe	ZWE	Unmet need for contraception (% of married wom...	SP.UWT.TFRT	NaN	NaN	NaN
16968	Zimbabwe	ZWE	Voice and Accountability: Estimate	VA.EST	NaN	NaN	NaN

16969 rows × 68 columns

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 [6]:
```
vdem = vdem[['country_text_id', 'country_name', 'year', 'v2x_polyarchy', 'v2peedueq
vdem
```

Out[6]:

	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 [7]:
```
vdem = vdem.query('year >= 1960 & year <= 2021')
vdem
```

Out[7]:

	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 [8]:

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

Out[8]:

	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 [9]:

```
vdem.sort_values(by=['country_name_vdem', 'year'], ascending=True)
vdem
```

Out[9]:

	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

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 [10]:

```
col19 = [x for x in wb.columns if x.startswith('19')]
col20 = [x for x in wb.columns if x.startswith('20')]
cols = ['Country Code', 'Country Name', 'Indicator Code'] + col19 + col20
wb = wb[cols]
wb
```

Out[10]:

	Country Code	Country Name	Indicator Code	1960	1961	1962	1963	
0	ARB	Arab World	EG.CFT.ACCS.ZS	NaN	NaN	NaN	NaN	
1	ARB	Arab World	EG.ELC.ACCS.ZS	NaN	NaN	NaN	NaN	
2	ARB	Arab World	NY.ADJ.DRES.GN.ZS	NaN	NaN	NaN	NaN	
3	ARB	Arab World	NY.ADJ.DFOR.GN.ZS	NaN	NaN	NaN	NaN	
4	ARB	Arab World	AG.LND.AGRI.ZS	NaN	30.981414	30.982663	31.007054	31.
...	...	...	...	...	...	...	...	
16964	ZWE	Zimbabwe	ER.PTD.TOTL.ZS	NaN	NaN	NaN	NaN	
16965	ZWE	Zimbabwe	AG.LND.FRLS.HA	NaN	NaN	NaN	NaN	
16966	ZWE	Zimbabwe	SL.UEM.TOTL.ZS	NaN	NaN	NaN	NaN	
16967	ZWE	Zimbabwe	SP.UWT.TFRT	NaN	NaN	NaN	NaN	
16968	ZWE	Zimbabwe	VA.EST	NaN	NaN	NaN	NaN	

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 [11]:

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

Out[11]:

	country_code	country_name_wb	feature	1960	1961	1962
0	ARB	Arab World	EG.CFT.ACCS.ZS	NaN	NaN	NaN
1	ARB	Arab World	EG.ELC.ACCS.ZS	NaN	NaN	NaN
2	ARB	Arab World	NY.ADJ.DRES.GN.ZS	NaN	NaN	NaN
3	ARB	Arab World	NY.ADJ.DFOR.GN.ZS	NaN	NaN	NaN
4	ARB	Arab World	AG.LND.AGRI.ZS	NaN	30.981414	30.982663
...	...	...	...	...	...	...
16964	ZWE	Zimbabwe	ER.PTD.TOTL.ZS	NaN	NaN	NaN
16965	ZWE	Zimbabwe	AG.LND.FRLS.HA	NaN	NaN	NaN
16966	ZWE	Zimbabwe	SL.UEM.TOTL.ZS	NaN	NaN	NaN
16967	ZWE	Zimbabwe	SP.UWT.TFRT	NaN	NaN	NaN
16968	ZWE	Zimbabwe	VA.EST	NaN	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 [12]:

```
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 [13]:
```
wb = wb.query('country_name_wb not in @noncountries')
wb
```

Out[13]:

	country_code	country_name_wb	feature	1960	1961	1962
3266	AFG	Afghanistan	EG.CFT.ACCS.ZS	NaN	NaN	NaN
3267	AFG	Afghanistan	EG.ELC.ACCS.ZS	NaN	NaN	NaN
3268	AFG	Afghanistan	NY.ADJ.DRES.GN.ZS	NaN	NaN	NaN
3269	AFG	Afghanistan	NY.ADJ.DFOR.GN.ZS	NaN	NaN	NaN
3270	AFG	Afghanistan	AG.LND.AGRI.ZS	NaN	57.878356	57.955016
...	...	...	...	...	...	...
16964	ZWE	Zimbabwe	ER.PTD.TOTL.ZS	NaN	NaN	NaN
16965	ZWE	Zimbabwe	AG.LND.FRLS.HA	NaN	NaN	NaN
16966	ZWE	Zimbabwe	SL.UEM.TOTL.ZS	NaN	NaN	NaN
16967	ZWE	Zimbabwe	SP.UWT.TFRT	NaN	NaN	NaN
16968	ZWE	Zimbabwe	VA.EST	NaN	NaN	NaN

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 [14]:
```
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_parliaments",
    "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_survey_mean_
    "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_rate",
    "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 [15]:

```
wb.feature = wb.feature.map(replace_map)
wb
```

```
C:\Users\brian\AppData\Local\Temp\ipykernel_12408\215342572.py:1: SettingWithCopyWar
ning:
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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  wb.feature = wb.feature.map(replace_map)
```

Out[15]:

	country_code	country_name_wb	feature	196
3266	AFG	Afghanistan	access_to_clean_fuels_and_technologies_for_coo...	Na
3267	AFG	Afghanistan	access_to_electricity	Na
3268	AFG	Afghanistan	natural_resources_depletion	Na
3269	AFG	Afghanistan	net_forest_depletion	Na
3270	AFG	Afghanistan	agricultural_land	Na
...	...	...	...	.
16964	ZWE	Zimbabwe	terrestrial_and_marine_protected_areas	Na
16965	ZWE	Zimbabwe	tree_cover_loss	Na
16966	ZWE	Zimbabwe	unemployment	Na
16967	ZWE	Zimbabwe	unmet_need_for_contraception	Na
16968	ZWE	Zimbabwe	voice_and_accountability	Na

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 [16]:
```
wb = pd.melt(wb, id_vars=['country_code', 'country_name_wb', 'feature'], value_vars
wb
```

Out[16]:

	country_code	country_name_wb	feature	var
0	AFG	Afghanistan	access_to_clean_fuels_and_technologies_for_coo...	
1	AFG	Afghanistan	access_to_electricity	
2	AFG	Afghanistan	natural_resources_depletion	
3	AFG	Afghanistan	net_forest_depletion	
4	AFG	Afghanistan	agricultural_land	
...	...	...	...	
808472	ZWE	Zimbabwe	terrestrial_and_marine_protected_areas	
808473	ZWE	Zimbabwe	tree_cover_loss	
808474	ZWE	Zimbabwe	unemployment	
808475	ZWE	Zimbabwe	unmet_need_for_contraception	
808476	ZWE	Zimbabwe	voice_and_accountability	

808477 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 [17]:
```
wb = wb.rename({'variable': 'year'}, axis=1)
wb=wb.pivot(index=['country_code', 'country_name_wb', 'year'], columns='feature', v
wb = pd.DataFrame(wb.to_records())
wb
```

Out[17]:

	country_code	country_name_wb	year	access_to_clean_fuels_and_technologies_for_co
0	AFG	Afghanistan	1960	
1	AFG	Afghanistan	1961	
2	AFG	Afghanistan	1962	
3	AFG	Afghanistan	1963	
4	AFG	Afghanistan	1964	
...	...	...	...	
11382	ZWE	Zimbabwe	2014	
11383	ZWE	Zimbabwe	2015	
11384	ZWE	Zimbabwe	2016	
11385	ZWE	Zimbabwe	2017	
11386	ZWE	Zimbabwe	2018	

11387 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 [18]:

```
wb.year = wb.year.astype(int)
wb
```

Out[18]:

	country_code	country_name_wb	year	access_to_clean_fuels_and_technologies_for_co
0	AFG	Afghanistan	1960	
1	AFG	Afghanistan	1961	
2	AFG	Afghanistan	1962	
3	AFG	Afghanistan	1963	
4	AFG	Afghanistan	1964	
...	...	...	...	
11382	ZWE	Zimbabwe	2014	
11383	ZWE	Zimbabwe	2015	
11384	ZWE	Zimbabwe	2016	
11385	ZWE	Zimbabwe	2017	
11386	ZWE	Zimbabwe	2018	

11387 rows × 74 columns

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]

I expect this to be a one-to-many relationship. There is one country code that will map to many years.

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 [19]:
```
md = pd.merge(vdem, wb, on=['country_code', 'year'], how='outer', indicator='matche
md.query('matched != "both"')
```

Out[19]:

	country_code	country_name_vdem	year	democracy	educational_equality	country
59	AFG	Afghanistan	2019	0.353	-1.262	
60	AFG	Afghanistan	2020	0.356	-0.963	
61	AFG	Afghanistan	2021	0.158	-0.990	
121	AGO	Angola	2019	0.365	-1.354	
122	AGO	Angola	2020	0.349	-1.354	
...	...	...	...	...	...	
12293	ZZB	Zanzibar	2017	0.267	1.661	
12294	ZZB	Zanzibar	2018	0.268	1.486	
12295	ZZB	Zanzibar	2019	0.266	1.486	
12296	ZZB	Zanzibar	2020	0.258	1.427	
12297	ZZB	Zanzibar	2021	0.276	1.779	

2838 rows × 78 columns

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 [20]:
```
md['matched'].value_counts()
```

Out[20]:
```
matched
both           9460
right_only     1927
left_only       911
Name: count, dtype: int64
```

In [21]:
```
md.query('matched == "right_only"').groupby(['country_code','country_name_wb']).cou
```

Out[21]:

		country_name_vdem	year	democracy	educational_equali
country_code	**country_name_wb**				
AND	**Andorra**	0	59	0	
ARE	**United Arab Emirates**	0	11	0	
ARM	**Armenia**	0	30	0	
ATG	**Antigua and Barbuda**	0	59	0	
AZE	**Azerbaijan**	0	30	0	
BGD	**Bangladesh**	0	11	0	
BHS	**Bahamas, The**	0	59	0	
BIH	**Bosnia and Herzegovina**	0	32	0	
BLR	**Belarus**	0	30	0	
BLZ	**Belize**	0	59	0	
BRN	**Brunei Darussalam**	0	59	0	
CMR	**Cameroon**	0	1	0	
DMA	**Dominica**	0	59	0	
EST	**Estonia**	0	30	0	
FSM	**Micronesia, Fed. Sts.**	0	59	0	
GEO	**Georgia**	0	30	0	
GRD	**Grenada**	0	59	0	
HRV	**Croatia**	0	31	0	
KAZ	**Kazakhstan**	0	30	0	
KGZ	**Kyrgyz Republic**	0	30	0	
KIR	**Kiribati**	0	59	0	
KNA	**St. Kitts and Nevis**	0	59	0	
LCA	**St. Lucia**	0	59	0	
LIE	**Liechtenstein**	0	59	0	
LTU	**Lithuania**	0	30	0	
LVA	**Latvia**	0	30	0	

country_code	country_name_wb	country_name_vdem	year	democracy	educational_equali
MCO	Monaco	0	59	0	
MDA	Moldova	0	30	0	
MHL	Marshall Islands	0	59	0	
MKD	North Macedonia	0	31	0	
MNE	Montenegro	0	38	0	
NRU	Nauru	0	59	0	
PLW	Palau	0	59	0	
SMR	San Marino	0	59	0	
SSD	South Sudan	0	51	0	
SVK	Slovak Republic	0	33	0	
SVN	Slovenia	0	29	0	
TJK	Tajikistan	0	30	0	
TKM	Turkmenistan	0	30	0	
TON	Tonga	0	59	0	
TUV	Tuvalu	0	59	0	
UKR	Ukraine	0	30	0	
UZB	Uzbekistan	0	30	0	
VCT	St. Vincent and the Grenadines	0	59	0	
WSM	Samoa	0	59	0	

45 rows × 76 columns

In [22]:
```
md.query('matched == "left_only"').groupby(['country_code','country_name_vdem']).co
```

Out[22]:

country_code	country_name_vdem	year	democracy	educational_equality	country_name_w
AFG	Afghanistan	3	3	3	
AGO	Angola	3	3	3	
ALB	Albania	3	3	3	
ARE	United Arab Emirates	3	3	3	
ARG	Argentina	3	3	3	
...	...	...	...	...	
YMD	South Yemen	31	31	31	
ZAF	South Africa	3	3	3	
ZMB	Zambia	3	3	3	
ZWE	Zimbabwe	3	3	3	
ZZB	Zanzibar	62	62	62	

182 rows × 76 columns

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]

After attempting to write some code to look at this I just manually went through and took some notes on what I found. This is by no means an exhaustive list of the issues.

North Macedonia in both? same country code south sudan in both?

wb = vdem Kyrgyz Republic = kyrgyzstan Slovak Republic = Slovakia

vdem 195 countries - list has 182 rows... German Democratic Republic is east germany? Somaliland unrecognized county in the horn of africa. part of somalia? south yemen part of yemen? Taiwan - conflict with china turkey now turkiye kosovo is a developing country Republic of Vietnam is south vietnam a lot of countries list are officially "republic of..."

I noticed that the World Bank coutnry list hsa current countries with their official country name since they are an official organization. The Vdem dataset contains data from countries that no longer exist, prior country names or spellings, disputed countries and so on since they are simply collecting the data and with a change in a country name (or the merge of countries) the data might tell a different story as is good to not change things for some historical perspecitve on democracy.

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 [23]:
```
md_final = pd.merge(vdem, wb, on=['country_code', 'year'], how='inner')
md_final
```

Out[23]:

	country_code	country_name_vdem	year	democracy	educational_equality	country_
0	MEX	Mexico	1960	0.232	-1.438	
1	MEX	Mexico	1961	0.234	-1.438	
2	MEX	Mexico	1962	0.233	-1.438	
3	MEX	Mexico	1963	0.233	-1.438	
4	MEX	Mexico	1964	0.231	-1.438	
...	...	...	...	...	...	
9455	HUN	Hungary	2014	0.666	1.129	
9456	HUN	Hungary	2015	0.621	1.129	
9457	HUN	Hungary	2016	0.606	1.081	
9458	HUN	Hungary	2017	0.561	1.081	
9459	HUN	Hungary	2018	0.483	0.754	

9460 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 [24]:

```
md_final.groupby('country_name_vdem')[['democracy']].mean().sort_values('democracy'
```

Out[24]:

country_name_vdem	democracy
Denmark	0.910237
Sweden	0.889424
Germany	0.877593
Luxembourg	0.874932
Australia	0.871169
...	...
Eritrea	0.082136
Oman	0.058492
United Arab Emirates	0.036646
Qatar	0.023678
Saudi Arabia	0.014763

172 rows × 1 columns

Denmark has the highest average level of democratic quality, Saudi Arabia has the lowest.

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 [25]:
```
md_final = md_final.assign(educational_equality_status =
                           pd.cut(md_final.educational_equality,
                                  bins=[-4, -2, -0.5, 0.5, 1.5, 4],
                                  labels=('extremely unequal', 'very unequal', 'som
md_final.groupby('educational_equality_status')[['gini_index', 'poverty_headcount_r
```

```
C:\Users\brian\AppData\Local\Temp\ipykernel_12408\1790123662.py:5: FutureWarning: Th
e default of observed=False is deprecated and will be changed to True in a future ve
rsion of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  md_final.groupby('educational_equality_status')[['gini_index', 'poverty_headcount_
ratio_at_national_poverty_lines']].mean()
```

Out[25]:

educational_equality_status	gini_index	poverty_headcount_ratio_at_national_poverty_lines
extremely unequal	39.590909	64.750000
very unequal	46.313500	39.662011
somewhat unequal	43.427273	25.362245
relatively equal	37.478538	23.143229
equal	32.763590	17.572274