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

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.

Problem 0

Import the following libraries:

In [1]:
```
import numpy as np
import pandas as pd
```

Problem 1

In the first part of this lab, the goal is to merge data from the United Nations World Health Organization (https://www.who.int/who-un/en/) with data from the Varieties of Democracy Project (https://www.v-dem.net/en/). The UN-WHO studies health outcomes in a cross-national context, and V-Dem studies the quality of democracy as it changes across countries and over time. We would want to merge these two datasets together if we wanted to study whether democratic quality can predict health outcomes.

The UN data contains cross-national time series data from the United Nations and World Health Organization, and includes three features:

- The number of physicians per 1000 people
- The percent of the population that is malnourished
- Health expenditure per capita

The VDem data comes from the Varieties of Democracy project, which aims to measure the quality of democracy and the amount of corruption in different countries over time (https://www.v-dem.net/en/data/data-version-8/). This data file contains indices regarding a country's democractic quality, level of civil liberites, and corruption. It also contains a binary indicator that separates countries into democratic and nondemocratic states, and it includes a categorizaton of the corruption scale.

The URLs for the two datasets are:

In [2]:
```
undata_url = "https://github.com/jkropko/DS-6001/raw/master/localdata/UNdata.cs
VDem_url = "https://github.com/jkropko/DS-6001/raw/master/localdata/vdem.csv"
```

Part a

Load both CSV files. Make sure to check whether there are rows that should not be included in the dataframe, and whether there are missing codes that should be replaced with `NaN`. Fix these problems at the data loading stage, if you can. (Don't worry about column names or category labels yet.) Also, the UN data covers the years 1960-2014, and the VDem data covers the years 1960-2015. To make the timeframe match up, delete rows in the VDem data from 2015. (1 point)

In [3]:
```
unData = pd.read_csv(undata_url)
unData = unData.replace('..', np.nan)
unData.drop(unData.tail(5).index,
            inplace = True)
unData
```

Out[3]:

	Series Name	Series Code	Country Name	Country Code	1960 [YR1960]	1961 [YR1961]	19 [YR19
0	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Afghanistan	AFG	0.0348442494869232	NaN	N
1	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Albania	ALB	0.276291221380234	NaN	N
2	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Algeria	DZA	0.173148155212402	NaN	N
3	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	American Samoa	ASM	NaN	NaN	N
4	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Andorra	ADO	NaN	NaN	N
...	...	...	...	...	...	...	
769	Health expenditure per capita (current US$)	SH.XPD.PCAP	West Bank and Gaza	WBG	NaN	NaN	N
770	Health expenditure per capita (current US$)	SH.XPD.PCAP	World	WLD	NaN	NaN	N
771	Health expenditure per capita (current US$)	SH.XPD.PCAP	Yemen, Rep.	YEM	NaN	NaN	N
772	Health expenditure per capita (current US$)	SH.XPD.PCAP	Zambia	ZMB	NaN	NaN	N
773	Health expenditure per capita (current US$)	SH.XPD.PCAP	Zimbabwe	ZWE	NaN	NaN	N

774 rows × 60 columns

In [4]:
```
vdem = pd.read_csv(VDem_url)
vdem = vdem[vdem['year'] < 2015]
vdem
```

Out[4]:

	X1	country_name	country_id	country_text_id	year	historical_date	codingstart	ga
0	1	Mexico	3	MEX	1960	1960-01-01	1900	
1	2	Mexico	3	MEX	1961	1961-01-01	1900	
2	3	Mexico	3	MEX	1962	1962-01-01	1900	
3	4	Mexico	3	MEX	1963	1963-01-01	1900	
4	5	Mexico	3	MEX	1964	1964-01-01	1900	
...	...	...	...	...	...	...	...	
8529	8530	Hungary	210	HUN	2008	2008-01-01	1918	
8530	8531	Hungary	210	HUN	2009	2009-01-01	1918	
8531	8532	Hungary	210	HUN	2010	2010-01-01	1918	
8532	8533	Hungary	210	HUN	2011	2011-01-01	1918	
8533	8534	Hungary	210	HUN	2012	2012-01-01	1918	

8458 rows × 100 columns

Part b

The UN data contain certain rows that refer to groups of countries instead of to individual countries. Here's a list of these non-countries:

In [5]:
```
noncountries = ['Arab World',  'Caribbean small states',  'Central Europe and t
    'Early-demographic dividend',  'East Asia & Pacific', 'East Asia & Pacific
    'East Asia & Pacific (IDA & IBRD countries)', 'Euro area', 'Europe & Centra
    'Europe & Central Asia (excluding high income)', 'Europe & Central Asia (II
    'Fragile and conflict affected situations', 'Heavily indebted poor countrie
    'High income', 'Late-demographic dividend', 'Latin America & Caribbean',
    'Latin America & Caribbean (excluding high income)',
    'Latin America & the Caribbean (IDA & IBRD countries)', 'Least developed co
    'Low & middle income', 'Low income', 'Lower middle income',
    'Middle East & North Africa', 'Middle East & North Africa (excluding high i
    'Middle East & North Africa (IDA & IBRD countries)',
    'Middle income', 'North America', 'OECD members',
    'Other small states', 'Pacific island small states', 'Post-demographic divi
    'Pre-demographic dividend', 'Small states', 'South Asia',
    'South Asia (IDA & IBRD)', 'Sub-Saharan Africa', 'Sub-Saharan Africa (exclu
    'Sub-Saharan Africa (IDA & IBRD countries)', 'Upper middle income', 'World'
```

We can use `.query()` to remove the non-countries from the data, but in this case there are complications due to the space in the name of the column `Country Name` and the use of an external list. So here let's use an alternative method:

First, apply the `.isin(noncountries)` method to the `Country Name` column of the UN data to create a series of values that are `True` if the `Country Name` on a row is one of the non-countries, and `False` otherwise. Second, use the `~` operator to negate the logical values: turn `True` to `False` and vice versa. Finally, pass this logical series to the

`.loc[]` attribute of the dataframe to drop the rows that refer to these noncountries from the UN data. (1 point)

(If you wanted to use `.query()`, you would first need to rename `Country Name` to remove the space, then you can use an `@` in front of `noncountries` to refer to the external list. But for this problem follow the instructions listed above.)

In [6]:
```
nons = unData['Country Name'].isin(noncountries)
unData = unData.loc[~nons]
unData
```

Out[6]:

	Series Name	Series Code	Country Name	Country Code	1960 [YR1960]	1961 [YR1961]	19 [YR19
0	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Afghanistan	AFG	0.0348442494869232	NaN	N
1	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Albania	ALB	0.276291221380234	NaN	N
2	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Algeria	DZA	0.173148155212402	NaN	N
3	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	American Samoa	ASM	NaN	NaN	N
4	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Andorra	ADO	NaN	NaN	N
...	...	...	...	...	...	...	
768	Health expenditure per capita (current US$)	SH.XPD.PCAP	Virgin Islands (U.S.)	VIR	NaN	NaN	N
769	Health expenditure per capita (current US$)	SH.XPD.PCAP	West Bank and Gaza	WBG	NaN	NaN	N
771	Health expenditure per capita (current US$)	SH.XPD.PCAP	Yemen, Rep.	YEM	NaN	NaN	N
772	Health expenditure per capita (current US$)	SH.XPD.PCAP	Zambia	ZMB	NaN	NaN	N
773	Health expenditure per capita (current US$)	SH.XPD.PCAP	Zimbabwe	ZWE	NaN	NaN	N

651 rows × 60 columns

Part c

Reshape the UN data to move the years from the columns to the rows. (Once the years are in the rows, they will have values such as "1960 [YR1960]".) (2 points)

In [7]:
```
unData = pd.melt(unData,
                 id_vars= ['Series Name', 'Series Code', 'Country Name', 'Count
                 value_vars= [f'{i} [YR{i}]' for i in range(1960, 2015)])
unData
```

Out[7]:

	Series Name	Series Code	Country Name	Country Code	variable	value
0	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Afghanistan	AFG	1960 [YR1960]	0.0348442494869232
1	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Albania	ALB	1960 [YR1960]	0.276291221380234
2	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Algeria	DZA	1960 [YR1960]	0.173148155212402
3	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	American Samoa	ASM	1960 [YR1960]	NaN
4	Physicians (per 1,000 people)	SH.MED.PHYS.ZS	Andorra	ADO	1960 [YR1960]	NaN
...	...	...	...	...	...	...
35800	Health expenditure per capita (current US$)	SH.XPD.PCAP	Virgin Islands (U.S.)	VIR	2014 [YR2014]	NaN
35801	Health expenditure per capita (current US$)	SH.XPD.PCAP	West Bank and Gaza	WBG	2014 [YR2014]	NaN
35802	Health expenditure per capita (current US$)	SH.XPD.PCAP	Yemen, Rep.	YEM	2014 [YR2014]	79.93696624
35803	Health expenditure per capita (current US$)	SH.XPD.PCAP	Zambia	ZMB	2014 [YR2014]	85.85307416
35804	Health expenditure per capita (current US$)	SH.XPD.PCAP	Zimbabwe	ZWE	2014 [YR2014]	57.71045218

35805 rows × 6 columns

Part d

Rename the `variable` column to `year`. Then use string methods to remove the ends such as "[YR1960]" from the values of the new `year` column and convert the column to an integer data type.

Also, for whatever reason, real world data often contains multiple variables that are just different representations of the same information. In this case, the `Series Name` and `Series Code` variables tell us exactly the same thing, and the `Country Name` and `Country Code` variables tell us exactly the same thing. Unless I have a very good reason to keep both, I generally prefer to drop variables that are redundant and coded in a less helpful way. So drop `Series Code` and `Country Code`. (2 points)

In [8]:
```
unData = unData.rename({'variable': 'year'}, axis=1)
unData['year'] = unData.year.str[0:5].astype(int)
unData = unData.drop(['Series Code', 'Country Code'], axis=1)
unData
```

Out[8]:

	Series Name	Country Name	year	value
0	Physicians (per 1,000 people)	Afghanistan	1960	0.0348442494869232
1	Physicians (per 1,000 people)	Albania	1960	0.276291221380234
2	Physicians (per 1,000 people)	Algeria	1960	0.173148155212402
3	Physicians (per 1,000 people)	American Samoa	1960	NaN
4	Physicians (per 1,000 people)	Andorra	1960	NaN
...	...	...	...	...
35800	Health expenditure per capita (current US$)	Virgin Islands (U.S.)	2014	NaN
35801	Health expenditure per capita (current US$)	West Bank and Gaza	2014	NaN
35802	Health expenditure per capita (current US$)	Yemen, Rep.	2014	79.93696624
35803	Health expenditure per capita (current US$)	Zambia	2014	85.85307416
35804	Health expenditure per capita (current US$)	Zimbabwe	2014	57.71045218

35805 rows × 4 columns

Part e

Reshape the data to move the values of `Series Name` to separate columns. Make sure all of the columns exist in the dataframe after reshaping and are not stored in a row index or multi-index. Then rename the columns so that all of the columns have concise and descriptive names. (2 points)

In [9]:
```
unData = pd.DataFrame(unData.pivot_table(index=['Country Name', 'year'],
                                         columns='Series Name',
```

```
                                        values='value').to_records())
unData
```

Out[9]:

	Country Name	year	Health expenditure per capita (current US$)	Physicians (per 1,000 people)	Prevalence of undernourishment (% of population)
0	Afghanistan	1960	NaN	0.034844	NaN
1	Afghanistan	1965	NaN	0.063428	NaN
2	Afghanistan	1970	NaN	0.064900	NaN
3	Afghanistan	1981	NaN	0.077000	NaN
4	Afghanistan	1986	NaN	0.183100	NaN
...	...	...	...	...	...
6281	Zimbabwe	2010	36.362794	0.068000	34.7
6282	Zimbabwe	2011	48.469580	0.083000	33.5
6283	Zimbabwe	2012	57.253763	NaN	33.2
6284	Zimbabwe	2013	62.309228	NaN	33.5
6285	Zimbabwe	2014	57.710452	NaN	34.0

6286 rows × 5 columns

In [10]:

```
unData = unData.rename({'Health expenditure per capita (current US$)': 'Health
                        'Physicians (per 1,000 people)': 'Physicians per 1,000
                        'Prevalence of undernourishment (% of population)': '%
                       axis=1)
unData
```

Out[10]:

	Country Name	year	Health expenditure per capita (USD)	Physicians per 1,000 people	% Undernourished
0	Afghanistan	1960	NaN	0.034844	NaN
1	Afghanistan	1965	NaN	0.063428	NaN
2	Afghanistan	1970	NaN	0.064900	NaN
3	Afghanistan	1981	NaN	0.077000	NaN
4	Afghanistan	1986	NaN	0.183100	NaN
...	...	...	...	...	...
6281	Zimbabwe	2010	36.362794	0.068000	34.7
6282	Zimbabwe	2011	48.469580	0.083000	33.5
6283	Zimbabwe	2012	57.253763	NaN	33.2
6284	Zimbabwe	2013	62.309228	NaN	33.5
6285	Zimbabwe	2014	57.710452	NaN	34.0

6286 rows × 5 columns

Part f

Next we are going to join the cleaned UN data with the VDem data. In a perfect world, both datasets would include a shared numeric country ID field that we can use to match countries in one dataset to countries in the other. Unfortunately the UN data identifies the countries only by name. Worse still, while there is a big overlap the two datasets cover different sets of countries.

First decide whether this merge is a one-to-one, one-to-many, many-to-one, or many-to-many merge and describe your rationale in words.

Then perform a test merge that checks whether your expectation that the merge is one-to-one, one-to-many, many-to-one, or many-to-many is confirmed, and reports whether each row is matched, appears only in the UN data, or appears only in the VDem data. Use the `.unique()` or `.value_counts()` method to display the names of the countries that are not matched. (2 points)

The merge between the UN data and the VDem data is a one-to-one merge because each country appears multiple times in each dataset (for many different years). This means that joining on the Country name condition is many to many.

In [11]:
```
merged_test = pd.merge(unData, vdem, left_on=['Country Name', 'year'], right_or
                       how='outer', indicator='matched',
                      validate='one_to_one')

merged_test['matched'].value_counts()
```

Out[11]:
```
both          4606
right_only    3852
left_only     1680
Name: matched, dtype: int64
```

In [12]:
```
merged_test.query("matched!='both'")['Country Name'].unique()
```

Out[12]:
```
array(['Albania', 'American Samoa', 'Andorra', 'Angola',
       'Antigua and Barbuda', 'Armenia', 'Aruba', 'Austria', 'Azerbaijan',
       'Bahamas, The', 'Bahrain', 'Bangladesh', 'Belarus', 'Belize',
       'Bermuda', 'Bosnia and Herzegovina', 'Brunei Darussalam',
       'Cabo Verde', 'Cameroon', 'Cayman Islands',
       'Central African Republic', 'Chad', 'Channel Islands', 'Comoros',
       'Congo, Dem. Rep.', 'Congo, Rep.', "Cote d'Ivoire", 'Croatia',
       'Cyprus', 'Czech Republic', 'Djibouti', 'Dominica', 'Ecuador',
       'Egypt, Arab Rep.', 'Equatorial Guinea', 'Estonia', 'France',
       'French Polynesia', 'Gabon', 'Gambia, The', 'Georgia', 'Greece',
       'Greenland', 'Grenada', 'Guam', 'Guatemala', 'Guinea',
       'Guinea-Bissau', 'Haiti', 'Honduras', 'Hong Kong SAR, China',
       'Hungary', 'Iceland', 'Iran, Islamic Rep.', 'Ireland', 'Israel',
       'Italy', 'Jamaica', 'Kazakhstan', 'Kiribati',
       'Korea, Dem. People's Rep.', 'Korea, Rep.', 'Kuwait',
       'Kyrgyz Republic', 'Lao PDR', 'Latvia', 'Lesotho', 'Liberia',
       'Lithuania', 'Luxembourg', 'Macao SAR, China', 'Macedonia, FYR',
       'Madagascar', 'Malaysia', 'Mali', 'Malta', 'Marshall Islands',
       'Mauritania', 'Micronesia, Fed. Sts.', 'Moldova', 'Monaco',
       'Montenegro', 'Myanmar', 'Nauru', 'New Caledonia', 'New Zealand',
       'Nicaragua', 'Niger', 'Northern Mariana Islands', 'Oman', 'Palau',
       'Panama', 'Puerto Rico', 'Russian Federation', 'Samoa',
       'San Marino', 'Sao Tome and Principe', 'Saudi Arabia', 'Senegal',
       'Serbia', 'Seychelles', 'Sierra Leone', 'Singapore',
       'Slovak Republic', 'Slovenia', 'St. Kitts and Nevis', 'St. Lucia',
       'St. Vincent and the Grenadines', 'Swaziland',
       'Syrian Arab Republic', 'Tajikistan', 'Timor-Leste', 'Togo',
       'Tonga', 'Trinidad and Tobago', 'Turkmenistan', 'Tuvalu',
       'Ukraine', 'United Arab Emirates', 'United Kingdom', 'Uzbekistan',
       'Venezuela, RB', 'Vietnam', 'Virgin Islands (U.S.)',
       'West Bank and Gaza', 'Yemen, Rep.', nan], dtype=object)
```

In [13]:
```
merged_test.query("matched!='both'")['country_name'].unique()
```

Out[13]:
```
array([nan, 'Mexico', 'Suriname', 'Sweden', 'Ghana', 'South Africa',
       'Japan', 'Burma_Myanmar', 'Russia', 'Albania', 'Egypt', 'Yemen',
       'Colombia', 'Brazil', 'United States', 'El Salvador',
       'South Yemen', 'Bangladesh', 'Bolivia', 'Haiti', 'Honduras',
       'Mali', 'Pakistan', 'Peru', 'Senegal', 'South Sudan', 'Sudan',
       'Vietnam_Democratic Republic of', 'Vietnam_Republic of',
       'Afghanistan', 'Argentina', 'Ethiopia', 'India', 'Kenya',
       'Korea_North', 'Korea_South', 'Kosovo', 'Lebanon', 'Nigeria',
       'Philippines', 'Tanzania', 'Taiwan', 'Thailand', 'Uganda',
       'Venezuela', 'Benin', 'Bhutan', 'Burkina Faso', 'Cambodia',
       'Indonesia', 'Mozambique', 'Nepal', 'Nicaragua', 'Niger', 'Zambia',
       'Zimbabwe', 'Guinea', 'Ivory Coast', 'Mauritania', 'Canada',
       'Australia', 'Botswana', 'Burundi', 'Cape Verde',
       'Central African Republic', 'Chile', 'Costa Rica', 'East Timor',
       'Ecuador', 'France', 'Germany', 'Guatemala', 'Iran', 'Iraq',
       'Ireland', 'Italy', 'Jordan', 'Lesotho', 'Liberia', 'Malawi',
       'Maldives', 'Mongolia', 'Morocco', 'Netherlands', 'Panama',
       'Papua New Guinea', 'Qatar', 'Sierra Leone', 'Spain', 'Syria',
       'Tunisia', 'Uruguay', 'Algeria', 'Angola', 'Cameroon', 'Chad',
       'China', 'Congo_Democratic Republic of', 'Congo_Republic of the',
       'Djibouti', 'Dominican Republic', 'Eritrea', 'Gabon', 'Gambia',
       'Guinea-Bissau', 'Jamaica', 'Kyrgyzstan', 'Laos', 'Libya',
       'Madagascar', 'Namibia', 'Palestine_West_Bank', 'Rwanda',
       'Somalia', 'Sri Lanka', 'Swaziland', 'Togo', 'Trinidad and Tobago',
       'German Democratic Republic', 'Palestine_Gaza', 'Somaliland',
       'Barbados', 'Belgium', 'Bosnia and Herzegovina', 'Bulgaria',
       'Comoros', 'Cuba', 'Cyprus', 'Czech Republic', 'Denmark', 'Fiji',
       'Finland', 'Guyana', 'Israel', 'Macedonia', 'Malaysia',
       'Mauritius', 'New Zealand', 'Norway', 'Paraguay', 'Romania',
       'Sao Tome and Principe', 'Saudi Arabia', 'Serbia', 'Seychelles',
       'Slovakia', 'Solomon Islands', 'Vanuatu'], dtype=object)
```

Part g

There are many unmatched rows in this merge. There are three reasons why rows failed to match:

- Differences in geographical coverage: for example, the VDem data includes Taiwan, but the UN data does not
- Differences in time coverage: for example, the UN data includes records for France every year from 1970 through 2014, and VDem includes rows for France from 1960 to 2012, leaving 12 rows for France without matching years
- Differences in spelling: for example, South Korea is called "Korea, Rep." in the UN data and "Korea_South" in the VDem data.

We can't do anything about differences in geographic or temporal coverage. But we can recode some country names to account for differences in spelling and to match more rows that should match. Here is a list of differently spelled countries:

- "Burma_Myanmar" in VDem is "Myanmar" in the UN data
- "Cape Verde" in VDem is "Cabo Verde" in the UN data
- "Congo_Democratic Republic of" in VDem is "Congo, Dem. Rep." in the UN data
- "Congo_Republic of the" in VDem is "Congo, Rep." in the UN data

- "East Timor" in VDem is "Timor-Leste" in the UN data
- "Egypt" in VDem is "Egypt, Arab Rep." in the UN data
- "Gambia" in VDem is "Gambia, The" in the UN data
- "Iran" in VDem is "Iran, Islamic Rep." in the UN data
- "Ivory Coast" in VDem is "Cote d'Ivoire" in the UN data
- "Korea_North" in VDem is "Korea, Dem. People's Rep." in the UN data
- "Korea_South" in VDem is "Korea, Rep." in the UN data
- "Kyrgyzstan" in VDem is "Kyrgyz Republic" in the UN data
- "Laos" in VDem is "Lao PDR" in the UN data
- "Macedonia" in VDem is "Macedonia, FYR" in the UN data
- "Palestine_West_Bank" in VDem is "West Bank and Gaza" in the UN Data (there is also "Palestine_Gaza" in VDem, but since the UN combines data for the West Bank and Gaza, let's just use "Palestine_West_Bank" for this assignment)
- "Russia" in VDem is "Russian Federation" in the UN data
- "Slovakia" in VDem is "Slovak Republic" in the UN data
- "Syria" in VDem is "Syrian Arab Republic" in the UN data
- "Venezuela" in VDem is "Venezuela, RB" in the UN data
- "Vietnam_Democratic Republic of" in VDem is "Vietnam" in the UN data
- "Yemen" in VDem is "Yemen, Rep." in the UN data

Recode the country names listed above in one of the two dataframes to match the names in the other dataframe. Then perform an inner join of the two dataframes. Some rows will be dropped because of differences in coverage, but no rows will be dropped because of differences in spelling. (2 points)

In [14]:
```
vdem.country_name = vdem.country_name.replace({
    "Burma_Myanmar":"Myanmar",
    "Cape Verde":"Cabo Verde",
    "Congo_Democratic Republic of":"Congo, Dem. Rep.",
    "Congo_Republic of the":"Congo, Rep.",
    "East Timor":"Timor-Leste",
    "Egypt":"Egypt, Arab Rep.",
    "Gambia":"Gambia, The",
    "Iran":"Iran, Islamic Rep.",
    "Ivory Coast":"Cote d'Ivoire",
    "Korea_North":"Korea, Dem. People's Rep.",
    "Korea_South":"Korea, Rep.",
    "Kyrgyzstan":"Kyrgyz Republic",
    "Laos":"Lao PDR",
    "Macedonia":"Macedonia, FYR",
    "Palestine_West_Bank":"West Bank and Gaza",
    "Russia":"Russian Federation",
    "Slovakia":"Slovak Republic",
    "Syria":"Syrian Arab Republic",
    "Venezuela":"Venezuela, RB",
    "Vietnam_Democratic Republic of":"Vietnam",
    "Yemen":"Yemen, Rep."})
```

In [15]:
```
merged_data = pd.merge(unData, vdem, left_on=['Country Name','year'], right_on=
merged_data
```

Out[15]:

	Country Name	year	Health expenditure per capita (USD)	Physicians per 1,000 people	% Undernourished	X1	country_name	countr
0	Afghanistan	1960	NaN	0.034844	NaN	1583	Afghanistan	
1	Afghanistan	1965	NaN	0.063428	NaN	1588	Afghanistan	
2	Afghanistan	1970	NaN	0.064900	NaN	1593	Afghanistan	
3	Afghanistan	1981	NaN	0.077000	NaN	1604	Afghanistan	
4	Afghanistan	1986	NaN	0.183100	NaN	1609	Afghanistan	
...	...	...	...	...	...	...	...	
5144	Zimbabwe	2010	36.362794	0.068000	34.7	3035	Zimbabwe	
5145	Zimbabwe	2011	48.469580	0.083000	33.5	3036	Zimbabwe	
5146	Zimbabwe	2012	57.253763	NaN	33.2	3037	Zimbabwe	
5147	Zimbabwe	2013	62.309228	NaN	33.5	3038	Zimbabwe	
5148	Zimbabwe	2014	57.710452	NaN	34.0	3039	Zimbabwe	

5149 rows × 104 columns

Problem 2

Kickstarter is a website in which people can pledge financial support for creative projects. Patrons are only charged if a project raises enough money to meet a pre-specified goal, and projects can offer items as "rewards" for patrons who contribute at particular levels. One interesting aspect of Kickstarter is the ability to search projects by "ending soon". If you have a few dollars to spare and want to feel like a hero, you can swoop in at the last minute to contribute enough for a project to meet its goal.

Cathie So created a project on Kaggle in which she scraped Kickstarter and collected data on 4000 live projects (projects that were currently collecting pledges from patrons) as of October 10, 2016, at 5pm Pacific time. The data are here:

In [16]:
```
kickstarter = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/localc
kickstarter
```

Out[16]:

	Unnamed: 0	amt.pledged	blurb	by	country	currency	end.time	l
0	0	15823.0	\n'Catalysts, Explorers & Secret Keepers: Wome...	Museum of Science Fiction	US	usd	2016-11-01T23:59:00-04:00	Wash
1	1	6859.0	\nA unique handmade picture book for kids & ar...	Tyrone Wells & Broken Eagle, LLC	US	usd	2016-11-25T01:13:33-05:00	P
2	2	17906.0	\nA horror comedy about a repairman who was in...	Tessa Stone	US	usd	2016-11-23T23:00:00-05:00	Ange
3	3	67081.0	\nThe Johnny Wander autobio omnibus you've all...	Johnny Wander	US	usd	2016-11-01T23:50:00-04:00	Br
4	4	32772.0	\nThe vision for this project is the establish...	Beau's All Natural Brewing Company	RW	cad	2016-11-18T23:05:48-05:00	
...	...	...	...	...	...	...	...	
3995	3995	4403.0	\nEARTH IS BUT ONE FRUIT ON THE TREE OF LIFE. ...	Lewis Brown	US	usd	2016-11-20T01:10:00-05:00	Den
3996	3996	1304.0	\nImagine designing an item with an easy-to-us...	Your Expressions	US	usd	2016-11-15T16:00:00-05:00	Fra
3997	3997	1.0	\nUnique themed London venue and hostel for 9g...	Martin Wojtala	GB	gbp	2016-10-30T09:36:06-04:00	Lond
3998	3998	10.0	\nAll in One Phone Case\n	All in One Phone Case	US	usd	2016-11-17T12:11:26-05:00	Talla
3999	3999	35.0	\nLuxury Sunglasses built with Titanium, Carbo...	Carlos Araujo	US	usd	2016-12-11T00:11:01-05:00	Ne

4000 rows × 13 columns

Part a

Notice that the `end.time` column, the date and time at which the project stops accepting pledges, is formatted as follows:

```
2016-11-01T23:59:00-04:00
```

This formatting is "YYYY-MM-DDThh:mm:ss-TZD": four digits for the year, a dash, two digits for the month, another dash, and two digits for the day; the "T" separates the dates from the time; two digits for the hour, minute and second, separated by colons; and the time zone expressed as hours difference from Greenwich mean time (also called UTC), and -04:00 is four hours earlier than UTC, for example.

But `end.time` is also currently read as a string, with `object` data type:

In [17]:
```
kickstarter.dtypes
```

Out[17]:
```
Unnamed: 0             int64
amt.pledged          float64
blurb                 object
by                    object
country               object
currency              object
end.time              object
location              object
percentage.funded      int64
state                 object
title                 object
type                  object
url                   object
dtype: object
```

Convert `end.time` to a timestamp, and extract the month, day, year, hour, minute, and second of the end time. To allow the `pd.to_datetime()` function to read timezones, use the `utc=True` argument. (2 points)

In [18]:
```
kickstarter['end.time'] = pd.to_datetime(kickstarter['end.time'], utc=True)
kickstarter['end.time']
```

Out[18]:
```
0      2016-11-02 03:59:00+00:00
1      2016-11-25 06:13:33+00:00
2      2016-11-24 04:00:00+00:00
3      2016-11-02 03:50:00+00:00
4      2016-11-19 04:05:48+00:00
                  ...
3995   2016-11-20 06:10:00+00:00
3996   2016-11-15 21:00:00+00:00
3997   2016-10-30 13:36:06+00:00
3998   2016-11-17 17:11:26+00:00
3999   2016-12-11 05:11:01+00:00
Name: end.time, Length: 4000, dtype: datetime64[ns, UTC]
```

In [19]:
```
kickstarter['month'] = [x.month for x in kickstarter['end.time']]
kickstarter['day'] = [x.day for x in kickstarter['end.time']]
kickstarter['year'] = [x.year for x in kickstarter['end.time']]
kickstarter['hour'] = [x.hour for x in kickstarter['end.time']]
kickstarter['minute'] = [x.minute for x in kickstarter['end.time']]
kickstarter['second'] = [x.second for x in kickstarter['end.time']]
kickstarter
```

Out[19]:

	Unnamed: 0	amt.pledged	blurb	by	country	currency	end.time	
0	0	15823.0	\n'Catalysts, Explorers & Secret Keepers: Wome...	Museum of Science Fiction	US	usd	2016-11-02 03:59:00+00:00	Wa
1	1	6859.0	\nA unique handmade picture book for kids & ar...	Tyrone Wells & Broken Eagle, LLC	US	usd	2016-11-25 06:13:33+00:00	
2	2	17906.0	\nA horror comedy about a repairman who was in...	Tessa Stone	US	usd	2016-11-24 04:00:00+00:00	Ar
3	3	67081.0	\nThe Johnny Wander autobio omnibus you've all...	Johnny Wander	US	usd	2016-11-02 03:50:00+00:00	
4	4	32772.0	\nThe vision for this project is the establish...	Beau's All Natural Brewing Company	RW	cad	2016-11-19 04:05:48+00:00	
...	...	...	...	...	...	...	...	
3995	3995	4403.0	\nEARTH IS BUT ONE FRUIT ON THE TREE OF LIFE. ...	Lewis Brown	US	usd	2016-11-20 06:10:00+00:00	D
3996	3996	1304.0	\nImagine designing an item with an easy-to-us...	Your Expressions	US	usd	2016-11-15 21:00:00+00:00	I
3997	3997	1.0	\nUnique themed London venue and hostel for 9g...	Martin Wojtala	GB	gbp	2016-10-30 13:36:06+00:00	Lo
3998	3998	10.0	\nAll in One Phone Case\n	All in One Phone Case	US	usd	2016-11-17 17:11:26+00:00	Ta
3999	3999	35.0	\nLuxury Sunglasses built with Titanium, Carbo...	Carlos Araujo	US	usd	2016-12-11 05:11:01+00:00	

4000 rows × 19 columns

Part b

Create a dataframe with one row for every ending day in the `kickstarter` data that reports the average amount pledged (`amt.pledged`) on each day. Sort the rows in descending order by average amount pledged, and display the five days with the highest averages. (2 points)

In [20]:
```
avg_pleged= pd.DataFrame(kickstarter.groupby(['month','day'])['amt.pledged'].me
avg_pleged = pd.DataFrame(avg_pleged.to_records()).sort_values(by='amt.pledged'
avg_pleged.head(5)
```

Out[20]:

	month	day	amt.pledged
46	12	14	47938.375000
6	11	4	26975.388889
13	11	11	24990.669065
49	12	17	22160.230769
20	11	18	21016.234043

Part c

Display the text of the longest `blurb` in the data. (2 points)

In [21]:
```
kickstarter['blurb'][kickstarter['blurb'].str.len().idxmax()]
```

Out[21]:
```
'\nA unique handmade picture book for kids & art lovers about a nervous monste
r who finds his courage with the help of a brave little girl\n'
```

Part d

How many blurbs for projects with end dates between November 15, 2016 and December 7, 2016 contain the phrase "science fiction"? [Hint: Don't forget to make this search case-insensitive and to sort the `kickstarter` dataframe by `end.time` before setting `end.time` as the index.] (2 points)

In [22]:
```
ks_sorted = kickstarter.sort_values(by='end.time', ascending=True)
ks_sorted.index = ks_sorted['end.time']

timeframe = ks_sorted['11/15/2016':'12/7/2016']
```

In [23]:
```
sum(timeframe['blurb'].str.lower().str.contains('science fiction'))
```

Out[23]:
```
6
```

In [ ]:
```

```