More `pandas`: `DataFrames`
ISTA 131 Hw6, Due Thursday 10/24/2019 at 11:59 pm

Introduction. This homework is intended to give you more practice with `pandas` and its `DataFrame` data structure. It uses a [dataset](#) from [Kaggle](#), an awesome data science website.

Instructions. Create a module named `hw6.py`. Below is the spec for 6 functions. Implement them and upload your module to the D2L Assignments folder.

Testing. Download `hw6_test.py` and auxiliary files and put them in the same folder as your `hw6.py` module. Run it from the command line to see your current correctness score. Each of the 6 functions is worth 16.7% of your correctness score. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 131 Hw6`, and a brief summary of the module. Each function must contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code (not necessary on this assignment).

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html
https://www.kaggle.com/fernandol/countries-of-the-world
https://www.kaggle.com/
https://pandas.pydata.org/pandas-docs/stable/text.html
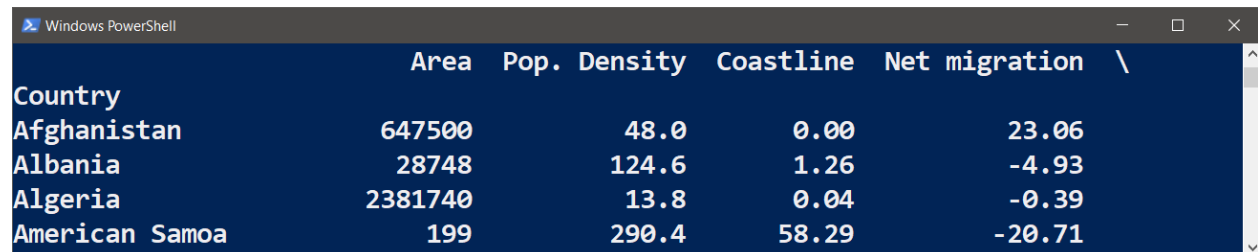
Function specifications.

`csv_to_dataframe:` This function takes a csv filename as an argument and returns a `DataFrame`. The csv looks like this (a screenshot of part of the file):

	A	B	C	D	E
1	Country	Region	Population	Area	Pop. Density
2	Afghanistan	ASIA (EX. NEAR E	31056997	647500	48,0
3	Albania	EASTERN EUROP	3581655	28748	124,6
4	Algeria	NORTHERN AFRI	32930091	2381740	13,8
5	American Sar	OCEANIA	57794	199	290,4

and the frame looks like this:

```
Windows PowerShell                                              —  □  ×
>>> df
                                    Region  Population  \
Country
Afghanistan             ASIA (EX. NEAR EAST)    31056997
Albania             EASTERN EUROPE              3581655
Algeria                     NORTHERN AFRICA     32930091
```

```
Windows PowerShell                                              —  □  ×
                  Area  Pop. Density  Coastline  Net migration  \
Country
Afghanistan      647500          48.0       0.00          23.06
Albania           28748         124.6       1.26          -4.93
Algeria         2381740          13.8       0.04          -0.39
American Samoa      199         290.4      58.29         -20.71
```

Compare the `Pop. Density` columns in the csv and the frame. Notice that the csv uses commas as decimal separators (European style), but the csv uses the periods you grew up with. Don't fix this yourself, make `read_csv` do it. There is an optional argument that will take care of it, you just have to find it in the docs or by using the `help` function.

`format_df:` This function takes a countries `DataFrame` as created by the previous function. Those regions look pretty nasty, so replace them with title-case versions of themselves, and with all leading and trailing whitespace stripped. Also, the country names have trailing whitespace. Replace the index by assigning a list of stripped country names to it. That will also get rid of the index name, `Country`. Alter the frame in-place. This link will be extremely helpful. It looks better now:

```
Windows PowerShell                                              —  □  ×
                      Region  Population      Area  Pop. Density  \
Afghanistan  Asia (Ex. Near East)    31056997    647500          48.0
Albania           Eastern Europe     3581655     28748         124.6
Algeria           Northern Africa    32930091   2381740         13.8
American Samoa            Oceania       57794       199         290.4
Andorra           Western Europe       71201       468         152.1
```
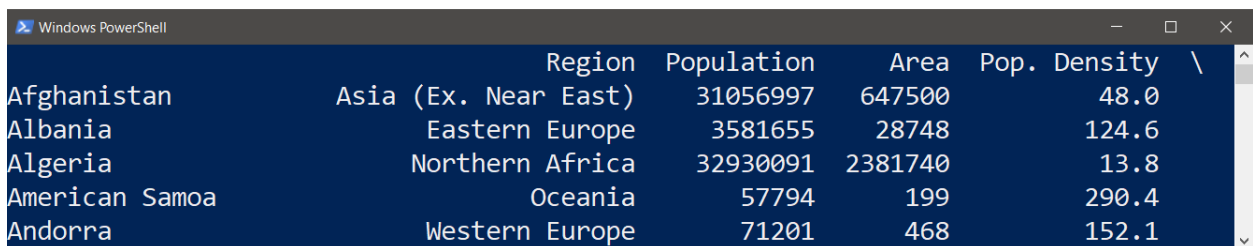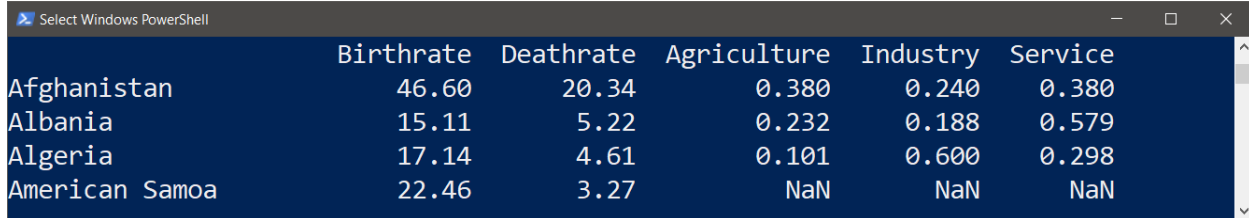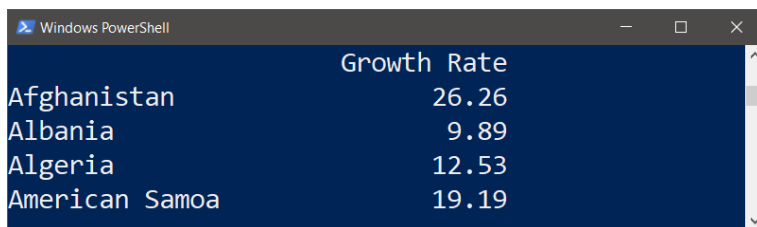
`growth_rate`: Now we are going to work with the frame's `Birthrate` and `Deathrate` data:

	Birthrate	Deathrate	Agriculture	Industry	Service
Afghanistan	46.60	20.34	0.380	0.240	0.380
Albania	15.11	5.22	0.232	0.188	0.579
Algeria	17.14	4.61	0.101	0.600	0.298
American Samoa	22.46	3.27	NaN	NaN	NaN

This function takes a formatted countries `DataFrame`. It adds a new column labeled `'Growth Rate'` to the frame. Each value in the `'Growth Rate'` column is calculated by subtracting the `Deathrate` for that row from the `Birthrate` for that row (we are ignoring the effects of migration). Alter the argument in-place, i.e. don't create a new frame. Here's what the new column looks like:
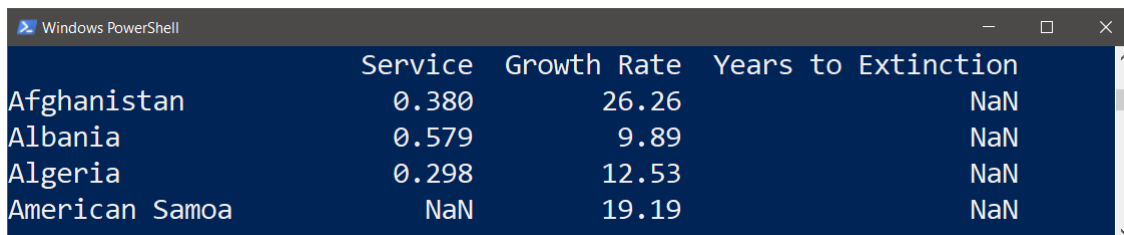
	Growth Rate
Afghanistan	26.26
Albania	9.89
Algeria	12.53
American Samoa	19.19

Add this code to your module:

```
def dod(p, r):
    num_yrs = 0
    while p > 2:
        p = p + p * r / 1000
        num_yrs += 1
    return num_yrs
```
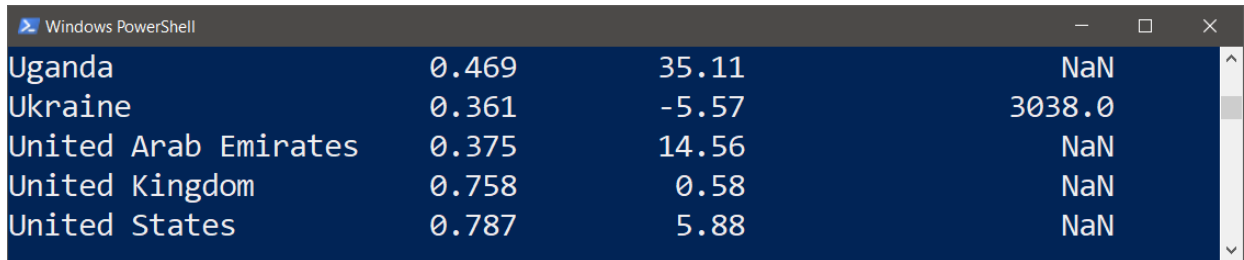
This function takes an initial population and a growth rate (**which must be negative – why?**) in 1000's of individuals per year and returns the number of years it will take for the population of the country to go extinct if the growth rate doesn't change. We consider the population extinct if it is down to no more than two individuals, but this stretches out the time considerably because of the way the math of exponential decay works. 1,000 or 10,000 individuals would probably be more reasonable definition of extinct.

`years_to_extinction`: This function takes a formatted countries `DataFrame` that has a `Growth Rate` column and adds a column labeled `'Years to Extinction'`. Initialize the values in this column to `np.nan`:
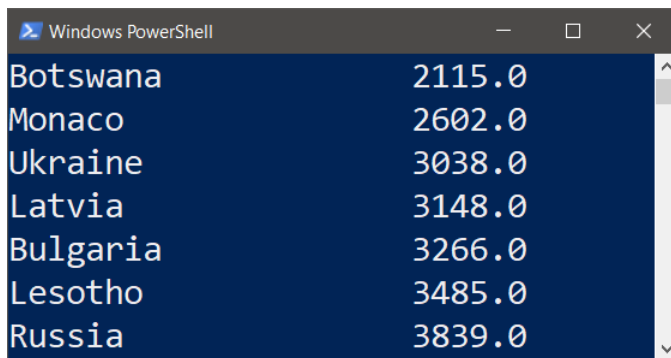
	Service	Growth Rate	Years to Extinction
Afghanistan	0.380	26.26	NaN
Albania	0.579	9.89	NaN
Algeria	0.298	12.53	NaN
American Samoa	NaN	19.19	NaN

Replace the `NaN` in the new column for every country that has a negative growth rate with the number of years until the population is extinct:

```
Windows PowerShell                                        —    □    ×
Uganda                    0.469        35.11              NaN
Ukraine                   0.361        -5.57           3038.0
United Arab Emirates      0.375        14.56              NaN
United Kingdom            0.758         0.58              NaN
United States             0.787         5.88              NaN
```
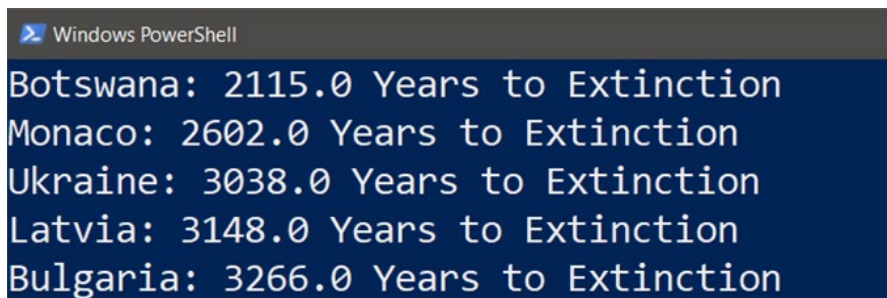
`dying_countries:` This function takes a formatted countries `DataFrame` that has a `Years to Extinction` column and returns a `Series` whose labels are the countries with negative growth rates and whose values are the number of years until they're dead in sorted order from first to last to die:

```
Windows PowerShell            —    □    ×
Botswana              2115.0
Monaco                2602.0
Ukraine               3038.0
Latvia                3148.0
Bulgaria              3266.0
Lesotho               3485.0
Russia                3839.0
```

`main:` `main` creates a frame from `countries_of_the_world.csv`, formats the frame, adds `Growth Rate` and `Years to Extinction` columns to it, and prints the top 5 dying countries in this format:

```
Windows PowerShell
Botswana: 2115.0 Years to Extinction
Monaco: 2602.0 Years to Extinction
Ukraine: 3038.0 Years to Extinction
Latvia: 3148.0 Years to Extinction
Bulgaria: 3266.0 Years to Extinction
```