assignment3

March 21, 2021

1 Assignment 3

All questions are weighted the same in this assignment. This assignment requires more individual learning then the last one did - you are encouraged to check out the pandas documentation to find functions or methods you might not have used yet, or ask questions on Stack Overflow and tag them as pandas and python related. All questions are worth the same number of points except question 1 which is worth 17% of the assignment grade.

Note: Questions 2-13 rely on your question 1 answer.

1.0.1 Question 1

Load the energy data from the file `assets/Energy Indicators.xls`, which is a list of indicators of energy supply and renewable electricity production from the United Nations for the year 2013, and should be put into a DataFrame with the variable name of **Energy**.

Keep in mind that this is an Excel file, and not a comma separated values file. Also, make sure to exclude the footer and header information from the datafile. The first two columns are unneccessary, so you should get rid of them, and you should change the column labels so that the columns are:

`['Country', 'Energy Supply', 'Energy Supply per Capita', '% Renewable]`

Convert `Energy Supply` to gigajoules (**Note: there are 1,000,000 gigajoules in a petajoule**). For all countries which have missing data (e.g. data with "...") make sure this is reflected as `np.NaN` values.

Rename the following list of countries (for use in later questions):

`"Republic of Korea": "South Korea", "United States of America": "United States", "United Kingdom of Great Britain and Northern Ireland": "United Kingdom", "China, Hong Kong Special Administrative Region": "Hong Kong"`

There are also several countries with numbers and/or parenthesis in their name. Be sure to remove these, e.g. `'Bolivia (Plurinational State of)'` should be `'Bolivia'`. `'Switzerland17'` should be `'Switzerland'`.

Next, load the GDP data from the file `assets/world_bank.csv`, which is a csv containing countries' GDP from 1960 to 2015 from World Bank. Call this DataFrame **GDP**.

Make sure to skip the header, and rename the following list of countries:

`"Korea, Rep.": "South Korea", "Iran, Islamic Rep.": "Iran", "Hong Kong SAR, China": "Hong Kong"`

Finally, load the Sciamgo Journal and Country Rank data for Energy Engineering and Power Technology from the file `assets/scimagojr-3.xlsx`, which ranks countries based on their journal contributions in the aforementioned area. Call this DataFrame **ScimEn**.

Join the three datasets: GDP, Energy, and ScimEn into a new dataset (using the intersection of country names). Use only the last 10 years (2006-2015) of GDP data and only the top 15 countries by Scimagojr 'Rank' (Rank 1 through 15).

The index of this DataFrame should be the name of the country, and the columns should be ['Rank', 'Documents', 'Citable documents', 'Citations', 'Self-citations', 'Citations per document', 'H index', 'Energy Supply', 'Energy Supply per Capita', '% Renewable', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015'].

This function should return a DataFrame with 20 columns and 15 entries, and the rows of the DataFrame should be sorted by "Rank".

[3]:
```
import pandas as pd
import numpy as np

def answer_one():

    Energy = pd.read_excel('assets/Energy Indicators.xls')#prepration of energy␣
↪indicators dataset

    # Filter all warnings. If you would like to see the warnings, please␣
↪comment the two lines below.
    import warnings
    warnings.filterwarnings('ignore')
    Energy = Energy.iloc[17:244]#remove unnecessary data for dataframe
    Energy['index'] = range(len(Energy))#create index and set it
    Energy.set_index('index', inplace = True)
    del Energy['Unnamed: 0']#delete unnecessary columns
    del Energy['Unnamed: 1']
    Energy.rename(columns = {'Unnamed: 2': 'Country', 'Unnamed: 3':␣
↪'energy_supply', 'Unnamed: 4': 'energy_supply_per_capita',
                             'Unnamed: 5': 'renewable_electricity_production'},␣
↪inplace = True)#change column names
    Energy.replace('...', np.NaN, inplace = True)#fill the missing values

    Energy['energy_supply'] = Energy['energy_supply'].apply(lambda x:␣
↪x*1000000)   #change petajoule to gigajoule

    import re

    #rename countries names and delete unwanted chars
    def rename_countries(country):

        country=re.sub(".\((.*)\)", "", country)#remove parantheses
        country=re.sub("\d*$", "", country)#remove un

        #to change specific country names
        specificcountry = ["Republic of Korea","United States of America",
```

```
                          "United Kingdom of Great Britain and Northern␣
↪Ireland",
                          "China, Hong Kong Special Administrative Region"]
       #use the common country names
       if country in specificcountry:
           if country == "Republic of Korea":
               return 'South Korea'
           elif country == "United States of America":
               return 'United States'
           elif country == "United Kingdom of Great Britain and Northern␣
↪Ireland":
               return 'United Kingdom'
           elif country == "China, Hong Kong Special Administrative Region":
               return 'Hong Kong'
       return country

   Energy['Country'] = Energy['Country'].apply(lambda x: rename_countries(x))

   GDP =pd.read_csv('assets/world_bank.csv')#prepration of world bank dataset
   GDP=GDP.iloc[3:]
   GDP.columns = GDP.iloc[0]
   GDP.drop([3], inplace = True)
   GDP.rename(columns = {"3": "", "Country Name" : "Country"}, inplace = True)
   GDP['index'] = range(len(GDP))#create index and set it
   GDP.set_index('index', inplace = True)
   GDP.head(5)

   def renameCountries(country):
       specifiCountryName = ["Korea, Rep." ,"Iran, Islamic Rep.", "Hong Kong␣
↪SAR, China"]

       if country in specifiCountryName:
           if country ==  "Korea, Rep.":
               return "South Korea"
           elif country == "Iran, Islamic Rep.":
               return "Iran"
           elif country == "Hong Kong SAR, China":
               return "Hong Kong"

       return country

   GDP['Country'] = GDP['Country'].apply(lambda x: renameCountries(x))#rename␣
↪some countries that have to change

   GDPQ1=GDP[["Country",'Country Code','Indicator Name','Indicator Code',#to␣
↪get specific years' datas
             2006,2007,2008,2009,2010,2011,2012,2013,2014,2015]]
```

```
    ScimEn = pd.read_excel("assets/scimagojr-3.xlsx")
    ScimEn = ScimEn.dropna()
    ScimEn = ScimEn[:15]#delete unwanted datas

    df=pd.merge(ScimEn, Energy, how = 'inner', on='Country')
    df=pd.merge(df, GDPQ1, how = 'inner', on = 'Country')
    df.rename(columns = {'energy_supply' : "Energy Supply",␣
↪"energy_supply_per_capita":"Energy Supply per Capita",
                  "renewable_electricity_production":"% Renewable", 2005:␣
↪'2005', 2006: '2006',2007: '2007',
                         2008:'2008', 2009:'2009', 2010:'2010', 2011:'2011',2012:
↪'2012',2013:'2013',2014:'2014',
                         2015:'2015'}, inplace = True)
    df.drop(['Indicator Name', 'Indicator Code', 'Country Code'], axis =␣
↪1,inplace = True)
    df.set_index('Country', inplace = True)
    #df.dropna(inplace = True)

    return df
```

[4]:
```
assert type(answer_one()) == pd.DataFrame, "Q1: You should return a DataFrame!"

assert answer_one().shape == (15,20), "Q1: Your DataFrame should have 20␣
↪columns and 15 entries!"
```

[5]:
```
# Cell for autograder.
```

1.0.2 Question 2

The previous question joined three datasets then reduced this to just the top 15 entries. When you joined the datasets, but before you reduced this to the top 15 items, how many entries did you lose?

This function should return a single number.

[6]:
```
%%HTML
<svg width="800" height="300">
  <circle cx="150" cy="180" r="80" fill-opacity="0.2" stroke="black"␣
↪stroke-width="2" fill="blue" />
  <circle cx="200" cy="100" r="80" fill-opacity="0.2" stroke="black"␣
↪stroke-width="2" fill="red" />
  <circle cx="100" cy="100" r="80" fill-opacity="0.2" stroke="black"␣
↪stroke-width="2" fill="green" />
  <line x1="150" y1="125" x2="300" y2="150" stroke="black" stroke-width="2"␣
↪fill="black" stroke-dasharray="5,3"/>
  <text x="300" y="165" font-family="Verdana" font-size="35">Everything but␣
↪this!</text>
</svg>
```

```
<IPython.core.display.HTML object>
```

[56]:
```
def answer_two():
    # Union A, B, C - Intersection A, B, C
    ScimEn = pd.read_excel("assets/scimagojr-3.xlsx")
    Energy = pd.read_excel('assets/Energy Indicators.xls')
    GDP = pd.read_csv('assets/world_bank.csv')
    #change column name to intersection and union
    Energy.rename(columns = {'Unnamed: 2': 'Country'}, inplace = True )
    GDP.rename(columns = {'Unnamed: 2': 'Country'}, inplace = True)

    union = pd.merge(pd.merge(Energy, GDP, on='Country', how='outer'), ScimEn,␣
 ↪on='Country', how='outer')
    intersect = pd.merge(pd.merge(Energy, GDP, on='Country'), ScimEn,␣
 ↪on='Country')



    return  len(union)-len(intersect)
    raise NotImplementedError()
```

[57]:
```
assert type(answer_two()) == int, "Q2: You should return an int number!"
```

1.0.3 Question 3

What are the top 15 countries for average GDP over the last 10 years?

This function should return a Series named `avgGDP` with 15 countries and their average GDP sorted in descending order.

[9]:
```
import numpy as np
def answer_three():
    top15 = answer_one()
    top15 = top15.iloc[:,10:]
    avgGDP = top15.mean(axis = 1)
    return avgGDP.sort_values(ascending = False)
    raise NotImplementedError()
```

[10]:
```
assert type(answer_three()) == pd.Series, "Q3: You should return a Series!"
```

1.0.4 Question 4

By how much had the GDP changed over the 10 year span for the country with the 6th largest average GDP?

This function should return a single number.

[76]:
```
def answer_four():
    top=answer_one()
    top["avg"] = answer_three()
    top.sort_values(by = ['avg'], ascending = False, inplace = True)
```

```
    sixth = top.iloc[5]
    last = sixth['2015']
    first = sixth['2006']
    return last-first
    raise NotImplementedError()
```

[77]:
```
# Cell for autograder.
```

1.0.5 Question 5

What is the mean energy supply per capita?

This function should return a single number.

[13]:
```
def answer_five():
    top=answer_one()
    perCapita=top['Energy Supply per Capita']
    mean=np.mean(perCapita)
    return mean
    raise NotImplementedError()
```

[14]:
```
# Cell for autograder.
```

1.0.6 Question 6

What country has the maximum % Renewable and what is the percentage?

This function should return a tuple with the name of the country and the percentage.

[15]:
```
def answer_six():
    top=answer_one()
    top= top.reset_index()
    top=top[['Country', '% Renewable']]
    index=top['% Renewable'].idxmax()
    return (top.iloc[index]['Country'], top.iloc[index]['% Renewable'])
    raise NotImplementedError()
```

[17]:
```
assert type(answer_six()) == tuple, "Q6: You should return a tuple!"

assert type(answer_six()[0]) == str, "Q6: The first element in your result␣
 ↪should be the name of the country!"
```

1.0.7 Question 7

Create a new column that is the ratio of Self-Citations to Total Citations. What is the maximum value for this new column, and what country has the highest ratio?

This function should return a tuple with the name of the country and the ratio.

[18]:
```
def answer_seven():
    def ratio(row):
        ratio=row['Self-citations']/row['Citations']
        return ratio
```

```
    top = answer_one()
    top = top.reset_index()
    top = top[['Country', 'Self-citations', 'Citations']]
    top['Self/Citations'] = top.apply(ratio, axis = 1)

    index=top['Self/Citations'].idxmax()

    return (top.iloc[index]['Country'], top.iloc[index]['Self/Citations'])
    raise NotImplementedError()
```

[19]:
```
assert type(answer_seven()) == tuple, "Q7: You should return a tuple!"

assert type(answer_seven()[0]) == str, "Q7: The first element in your result␣
↪should be the name of the country!"
```

1.0.8 Question 8

Create a column that estimates the population using Energy Supply and Energy Supply per capita. What is the third most populous country according to this estimate?

This function should return the name of the country

[22]:
```
def answer_eight():
    def estimate_population(row):
        population=row['Energy Supply']/row['Energy Supply per Capita']
        return population

    top = answer_one()
    top = top.reset_index()
    top = top[['Country', 'Energy Supply', 'Energy Supply per Capita']]
    top['population'] = top.apply(estimate_population, axis = 1)

    index=top['population'].idxmax()
    return top.iloc[index]['Country']
    raise NotImplementedError()
```

[23]:
```
assert type(answer_eight()) == str, "Q8: You should return the name of the␣
↪country!"
```

1.0.9 Question 9

Create a column that estimates the number of citable documents per person. What is the correlation between the number of citable documents per capita and the energy supply per capita? Use the `.corr()` method, (Pearson's correlation).

This function should return a single number.

(Optional: Use the built-in function `plot9()` to visualize the relationship between Energy Supply per Capita vs. Citable docs per Capita)

[24]:
```
def answer_nine():
    Top15 = answer_one()
    Top15['PopEst'] = Top15['Energy Supply'] / Top15['Energy Supply per Capita']
    Top15['Citable docs per Capita']=Top15['Citable documents'] /␣
 ↪Top15['PopEst']

    return Top15['Citable docs per Capita'].corr(Top15['Energy Supply per␣
 ↪Capita'])
    raise NotImplementedError()
```

[25]:
```
def plot9():
    import matplotlib as plt
    %matplotlib inline

    Top15 = answer_one()
    Top15['PopEst'] = Top15['Energy Supply'] / Top15['Energy Supply per Capita']
    Top15['Citable docs per Capita'] = Top15['Citable documents'] /␣
 ↪Top15['PopEst']
    Top15.plot(x='Citable docs per Capita', y='Energy Supply per Capita',␣
 ↪kind='scatter', xlim=[0, 0.0006])
```

[27]:
```
assert answer_nine() >= -1. and answer_nine() <= 1., "Q9: A valid correlation␣
 ↪should between -1 to 1!"
```

1.0.10 Question 10

Create a new column with a 1 if the country's % Renewable value is at or above the median for all countries in the top 15, and a 0 if the country's % Renewable value is below the median.

This function should return a series named `HighRenew` *whose index is the country name sorted in ascending order of rank.*

[28]:
```
def answer_ten():
    Top15 = answer_one()
    median = Top15['% Renewable'].median()
    Top15['HighRenew'] = [1 if x>= median else 0 for x in Top15['% Renewable']]
    return Top15['HighRenew']
    raise NotImplementedError()
```

[29]:
```
assert type(answer_ten()) == pd.Series, "Q10: You should return a Series!"
```

1.0.11 Question 11

Use the following dictionary to group the Countries by Continent, then create a DataFrame that displays the sample size (the number of countries in each continent bin), and the sum, mean, and std deviation for the estimated population of each country.

```
ContinentDict  = {'China':'Asia',
                  'United States':'North America',
                  'Japan':'Asia',
```

```
                'United Kingdom':'Europe',
                'Russian Federation':'Europe',
                'Canada':'North America',
                'Germany':'Europe',
                'India':'Asia',
                'France':'Europe',
                'South Korea':'Asia',
                'Italy':'Europe',
                'Spain':'Europe',
                'Iran':'Asia',
                'Australia':'Australia',
                'Brazil':'South America'}
```

This function should return a DataFrame with index named Continent `['Asia', 'Australia', 'Europe', 'North America', 'South America']` *and columns* `['size', 'sum', 'mean', 'std']`

[41]:
```
def answer_eleven():
    Top15 = answer_one()
    ContinentDict  = {'China':'Asia',
                  'United States':'North America',
                  'Japan':'Asia',
                  'United Kingdom':'Europe',
                  'Russian Federation':'Europe',
                  'Canada':'North America',
                  'Germany':'Europe',
                  'India':'Asia',
                  'France':'Europe',
                  'South Korea':'Asia',
                  'Italy':'Europe',
                  'Spain':'Europe',
                  'Iran':'Asia',
                  'Australia':'Australia',
                  'Brazil':'South America'}

    continets = pd.DataFrame(columns = ['size', 'sum', 'mean', 'std'])
    Top15['Estimate Population'] = Top15['Energy Supply'] / Top15['Energy␣
 ↪Supply per Capita']

    for group, frame in Top15.groupby(ContinentDict):
        continets.loc[group] = [len(frame), frame['Estimate Population'].sum(),
                                frame['Estimate Population'].
 ↪mean(),frame['Estimate Population'].std()]
    return continets

    raise NotImplementedError()
```

[42]:
```
assert type(answer_eleven()) == pd.DataFrame, "Q11: You should return a␣
 ↪DataFrame!"
```

```
assert answer_eleven().shape[0] == 5, "Q11: Wrong row numbers!"

assert answer_eleven().shape[1] == 4, "Q11: Wrong column numbers!"
```

1.0.12 Question 12

Cut % Renewable into 5 bins. Group Top15 by the Continent, as well as these new % Renewable bins. How many countries are in each of these groups?

This function should return a Series with a MultiIndex of `Continent`, then the bins for `% Renewable`. Do not include groups with no countries.

[50]:
```
def answer_twelve():
    Top15 = answer_one()
    ContinentDict  = {'China':'Asia',
                      'United States':'North America',
                      'Japan':'Asia',
                      'United Kingdom':'Europe',
                      'Russian Federation':'Europe',
                      'Canada':'North America',
                      'Germany':'Europe',
                      'India':'Asia',
                      'France':'Europe',
                      'South Korea':'Asia',
                      'Italy':'Europe',
                      'Spain':'Europe',
                      'Iran':'Asia',
                      'Australia':'Australia',
                      'Brazil':'South America'}

    Top15["bins"] = pd.cut(Top15["% Renewable"], 5)

    return Top15.groupby([ContinentDict, Top15['bins']]).size()
    raise NotImplementedError()
```

[51]:
```
assert type(answer_twelve()) == pd.Series, "Q12: You should return a Series!"

assert len(answer_twelve()) == 9, "Q12: Wrong result numbers!"
```

1.0.13 Question 13

Convert the Population Estimate series to a string with thousands separator (using commas). Use all significant digits (do not round the results).

e.g. 12345678.90 -> 12,345,678.90

This function should return a series `PopEst` whose index is the country name and whose values are the population estimate string

[ ]:
```
def answer_thirteen():
    Top15 = answer_one()
    Top15["Population"] = Top15['Energy Supply'] / Top15['Energy Supply per␣
 ↪Capita']
    return Top15['Population'].apply(lambda x: '{0:,}'.format(x))
    raise NotImplementedError()
```

[ ]:
```
assert type(answer_thirteen()) == pd.Series, "Q13: You should return a Series!"

assert len(answer_thirteen()) == 15, "Q13: Wrong result numbers!"
```

1.0.14 Optional

Use the built in function `plot_optional()` to see an example visualization.

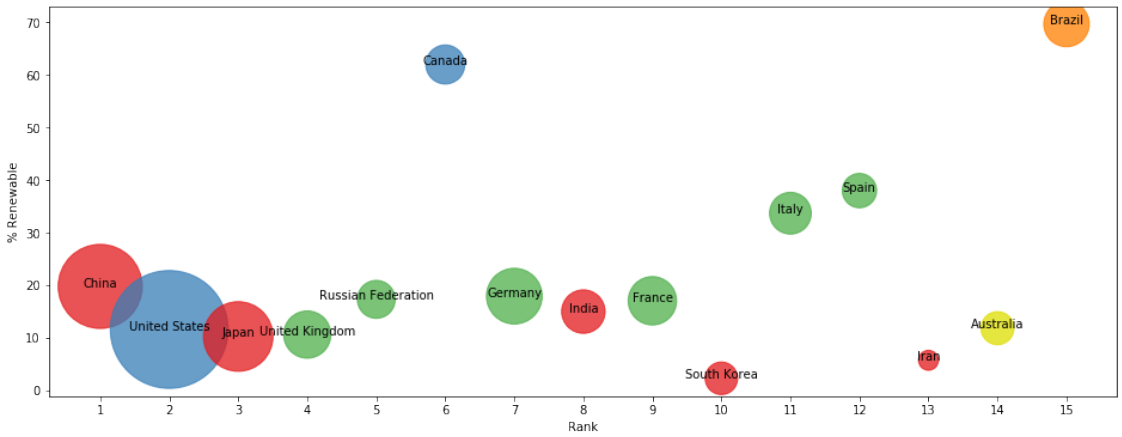
[106]:
```
def plot_optional():
    import matplotlib as plt
    %matplotlib inline
    Top15 = answer_one()
    ax = Top15.plot(x='Rank', y='% Renewable', kind='scatter',
                    ␣
 ↪c=['#e41a1c','#377eb8','#e41a1c','#4daf4a','#4daf4a','#377eb8','#4daf4a','#e41a1c',
                        ␣
 ↪'#4daf4a','#e41a1c','#4daf4a','#4daf4a','#e41a1c','#dede00','#ff7f00'],
                    xticks=range(1,16), s=6*Top15['2014']/10**10, alpha=.75,␣
 ↪figsize=[16,6]);

    for i, txt in enumerate(Top15.index):
        ax.annotate(txt, [Top15['Rank'][i], Top15['% Renewable'][i]],␣
 ↪ha='center')

    print("This is an example of a visualization that can be created to help␣
 ↪understand the data. \
This is a bubble chart showing % Renewable vs. Rank. The size of the bubble␣
 ↪corresponds to the countries' \
2014 GDP, and the color corresponds to the continent.")
```

[108]:
```
plot_optional()
```

```
This is an example of a visualization that can be created to help understand the
data. This is a bubble chart showing % Renewable vs. Rank. The size of the
bubble corresponds to the countries' 2014 GDP, and the color corresponds to the
continent.
```


Brazil
Canada
Spain
Italy
China
Russian Federation
Germany
India
France
United States
Japan
United Kingdom
Australia
Iran
South Korea
% Renewable
Rank
0
10
20
30
40
50
60
70
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15