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 (http://pandas.pydata.org/pandas-docs/stable/) to find functions or methods you might not have used yet, or ask questions on Stack Overflow (http://stackoverflow.com/) and tag them as pandas and python related. All questions are worth the same number of points except question 1 which is worth 20% of the assignment grade.

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

In [ ]:

```

```

In [1]:

```
import pandas as pd
import numpy as np
import re
import string as s
# Filter all warnings. If you would like to see the warnings, please comment the
two lines below.
import warnings
warnings.filterwarnings('ignore')
```

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 (assets/Energy%20Indicators.xls) from the United Nations (http://unstats.un.org/unsd/environment/excel_file_tables/2013/Energy%20Indicators.xls) 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 parenthesis in their name. Be sure to remove these, e.g. `'Bolivia (Plurinational State of)'` should be `'Bolivia'`.

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 (http://data.worldbank.org/indicator/NY.GDP.MKTP.CD). 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 (http://www.scimagojr.com/countryrank.php?category=2102) 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".

In [2]:

```
def simplifyName(country):
    x = country
    if '(' in x:
        x = x.split('(')[0].strip()
    digit = ""
    for char in x:
        if str.isdigit(char):
            digit = str(char);
            x = x.split(digit)[0].strip()
            break
        else:
            continue
    return x

def answer_one():
    # Read the excel file into the database, replace all ellipses to NaN values
    Energy = pd.read_excel("assets/Energy Indicators.xls", na_values = '...')[17
:246]
    #Drop unwanted columns
    Energy.drop(['Unnamed: 0', 'Unnamed: 1'], axis = 'columns', inplace = True)
    #Rename Column headers respectively.
    Energy.rename(columns={'Unnamed: 2': 'Country', 'Unnamed: 3':'Energy Supply'
,
                          'Unnamed: 4':'Energy Supply per Capita','Unnamed: 5':
'% Renewable'},inplace = True)
    #Convert Pentajoules to Gigajoules
    Energy['Energy Supply']*=10**6
    #Clean up the data in the country column ----------
    #Remove any parenthesis in the name of the country and any random numbers be
hind country name
    for index, row in Energy.iterrows():
        if (pd.isnull(row['Country'])):
            pass
        else:
            row['Country']= simplifyName(row['Country'])
    #Replace the name of the following countries:
    Energy = Energy.replace(["Republic of Korea", "United States of America",
                "United Kingdom of Great Britain and Northern Ireland",
                "China, Hong Kong Special Administrative Region"],
                   ["South Korea", "United States", "United Kingdom", "Hong Kon
g"])

    # Read the World Bank file by the world bank and manipulate it accordingly
 (LOL i give up commenting my actions)
    GDP = pd.read_csv("assets/world_bank.csv", skiprows = 4).drop(['Country Cod
e'],axis = 1).rename(
        columns={'Country Name': 'Country'}).replace(["Korea, Rep.", "Iran, Isla
mic Rep.","Hong Kong SAR, China"],
                                                      ["South Korea", "Iran", "Ho
ng Kong"])
    GDP = GDP[['Country','2006','2007','2008','2009','2010','2011','2012','2013'
,'2014','2015']]
    # Read the xlsx file and manipulate it accordingly.
    ScimEn = pd.read_excel("assets/scimagojr-3.xlsx")
    #Merge 3 dataframes, joining on country names
    result = pd.merge(pd.merge(ScimEn, Energy, how = "inner", on = 'Country'),GD
P, how = "inner", on = 'Country').set_index('Country')
    return result[result['Rank'].lt(16)]
```

```python
    raise NotImplementedError()
test = answer_one()
test
#test = test.set_index('Country')
#test.loc['South Korea']
```

Out[2]:

	Rank	Documents	Citable documents	Citations	Self-citations	Citations per document	H index	Energ Suppl
Country								
China	1	127050	126767	597237	411683	4.70	138	1.271910e+1
United States	2	96661	94747	792274	265436	8.20	230	9.083800e+1
Japan	3	30504	30287	223024	61554	7.31	134	1.898400e+1
United Kingdom	4	20944	20357	206091	37874	9.84	139	7.920000e+0
Russian Federation	5	18534	18301	34266	12422	1.85	57	3.070900e+1
Canada	6	17899	17620	215003	40930	12.01	149	1.043100e+1
Germany	7	17027	16831	140566	27426	8.26	126	1.326100e+1
India	8	15005	14841	128763	37209	8.58	115	3.319500e+1
France	9	13153	12973	130632	28601	9.93	114	1.059700e+1
South Korea	10	11983	11923	114675	22595	9.57	104	1.100700e+1
Italy	11	10964	10794	111850	26661	10.20	106	6.530000e+0
Spain	12	9428	9330	123336	23964	13.08	115	4.923000e+0
Iran	13	8896	8819	57470	19125	6.46	72	9.172000e+0
Australia	14	8831	8725	90765	15606	10.28	107	5.386000e+0
Brazil	15	8668	8596	60702	14396	7.00	86	1.214900e+1

In [3]:

```python
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!"
```

In [4]:

```python
# Cell for autograder.
```

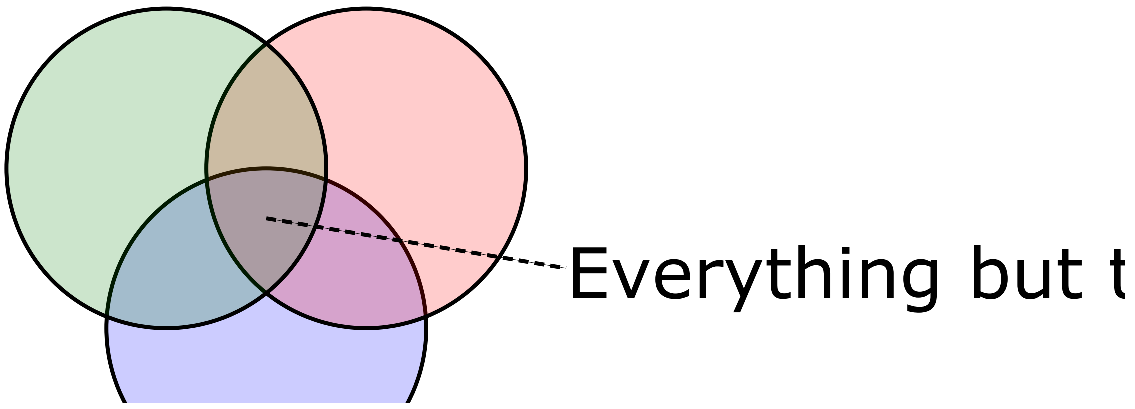
Question 2

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.

In [5]:

```
%%HTML
<svg width="800" height="300">
  <circle cx="150" cy="180" r="80" fill-opacity="0.2" stroke="black" stroke-widt
h="2" fill="blue" />
  <circle cx="200" cy="100" r="80" fill-opacity="0.2" stroke="black" stroke-widt
h="2" fill="red" />
  <circle cx="100" cy="100" r="80" fill-opacity="0.2" stroke="black" stroke-widt
h="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 thi
s!</text>
</svg>
```



In [53]:

```
def simplifyName(country):
    x = country
    if '(' in x:
        x = x.split('(')[0].strip()
    digit = ""
    for char in x:
        if str.isdigit(char):
            digit = str(char);
            x = x.split(digit)[0].strip()
            break
        else:
            continue
    return x
def answer_two():
    # YOUR CODE HERE
    # Read the excel file into the database, replace all ellipses to NaN values
    energy = pd.read_excel('assets/Energy Indicators.xls', skiprows=17,skip_foot
er= 38)
    energy = energy[['Unnamed: 1','Petajoules','Gigajoules','%']]
    energy.columns = ['Country', 'Energy Supply', 'Energy Supply per Capita', '%
Renewable']
    energy[['Energy Supply', 'Energy Supply per Capita', '% Renewable']] =  ener
gy[['Energy Supply', 'Energy Supply per Capita', '% Renewable']].replace('...',n
p.NaN).apply(pd.to_numeric)
    energy['Energy Supply'] = energy['Energy Supply']*1000000
    energy['Country'] = energy['Country'].replace({'China, Hong Kong Special Adm
inistrative Region':'Hong Kong','United Kingdom of Great Britain and Northern Ir
eland':'United Kingdom','Republic of Korea':'South Korea','United States of Amer
ica':'United States','Iran (Islamic Republic of)':'Iran'})
    energy['Country'] = energy['Country'].str.replace(" \(.*\)","")
    # Read the World Bank file by the world bank and manipulate it accordingly
 (LOL i give up commenting my actions)
    GDP = pd.read_csv("assets/world_bank.csv", skiprows = 4).drop(['Country Cod
e'],axis = 1).rename(
        columns={'Country Name': 'Country'}).replace(["Korea, Rep.", "Iran, Isla
mic Rep.","Hong Kong SAR, China"],
                                                    ["South Korea", "Iran", "Ho
ng Kong"])
    GDP = GDP[['Country','2006','2007','2008','2009','2010','2011','2012','2013'
,'2014','2015']]
    # Read the xlsx file and manipulate it accordingly.
    ScimEn = pd.read_excel("assets/scimagojr-3.xlsx")
    #Merge 3 dataframes, joining on country names
    result = pd.merge(pd.merge(ScimEn, energy, how = "inner", on = 'Country'),GD
P, how = "inner", on = 'Country').set_index('Country')
    result2 = pd.merge(pd.merge(ScimEn, energy, how = "outer", on = 'Country'),G
DP, how = "outer", on = 'Country').set_index('Country')
    ## Answer: Finding |(A n B n C)'|= |A u B u C| - |A n B n C|
    return (len(result2)-len(result))
    raise NotImplementedError()
test = answer_two()
test
```

Out[53]:

```
156
```

In [7]:

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

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.*

In [8]:

```
def answer_three():
    Top15 = answer_one()#a dataframe
    columns = ['2006','2007','2008','2009','2010','2011','2012','2013','2014','2
015']
    Top15['Mean'] = Top15[columns].mean(axis = 1)
    avgGDP = Top15.sort_values(by = 'Mean', ascending = False)['Mean']
    return avgGDP
    raise NotImplementedError()
test = answer_three()
test
```

Out[8]:

```
Country
United States         1.536434e+13
China                 6.348609e+12
Japan                 5.542208e+12
Germany               3.493025e+12
France                2.681725e+12
United Kingdom        2.487907e+12
Brazil                2.189794e+12
Italy                 2.120175e+12
India                 1.769297e+12
Canada                1.660647e+12
Russian Federation    1.565459e+12
Spain                 1.418078e+12
Australia             1.164043e+12
South Korea           1.106715e+12
Iran                  4.441558e+11
Name: Mean, dtype: float64
```

In [9]:

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

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.

In [10]:

```
def answer_four():
    # YOUR CODE HERE
    Top15 = answer_one()
    cty = answer_three().index[5]
    return (Top15['2015'][cty] - Top15['2006'][cty])

    raise NotImplementedError()
test =answer_four()
test
```

Out[10]:

```
246702696075.3999
```

In [11]:

```
# Cell for autograder.
```

Question 5

What is the mean energy supply per capita?

This function should return a single number.

In [12]:

```
def answer_five():
    # YOUR CODE HERE
    Top15 = answer_one()['Energy Supply per Capita'].rename(
        columns={'Energy Supply per Capita': 'Energy'}
    )
    return Top15.mean()
    raise NotImplementedError()
test = answer_five()
test
```

Out[12]:

```
157.6
```

In [13]:

```
# Cell for autograder.
```

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.

In [14]:

```
def answer_six():
    # YOUR CODE HERE
    Top15 = answer_one()['% Renewable']
    Top15 = Top15.sort_values(ascending = False)
    return (Top15.index[0],Top15[0])
    raise NotImplementedError()
test = answer_six()
test
```

Out[14]:

```
('Brazil', 69.64803)
```

In [15]:

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

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

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.

In [16]:

```
def answer_seven():
    # YOUR CODE HERE
    Top15 = answer_one()[['Self-citations','Citations']]
    Top15['ratio'] = Top15.apply(lambda row: row['Self-citations']/ row['Citatio
ns'], axis = 1)
    Top15.sort_values(by = 'ratio', ascending = False, inplace = True)
    return (Top15.index[0], Top15['ratio'][0])

    raise NotImplementedError()
answer_seven()
```

Out[16]:

```
('China', 0.6893126179389422)
```

In [17]:

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

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

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

In [18]:

```
def answer_eight():
    # YOUR CODE HERE
    top15 = answer_one()[['Energy Supply','Energy Supply per Capita']]
    top15['pop'] = top15.apply(lambda row: row['Energy Supply']/row['Energy Supp
ly per Capita'], axis = 1)
    top15.sort_values(by = 'pop', inplace = True, ascending = False)
    return (top15.index[2])
    raise NotImplementedError()
answer_eight()
```

Out[18]:

```
'United States'
```

In [19]:

```
assert type(answer_eight()) == str, "Q8: You should return the name of the count
ry!"
```

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)

In [20]:

```
def answer_nine():
    # YOUR CODE HERE
    import scipy.stats as stats
    top15 = answer_one()[['Energy Supply','Energy Supply per Capita','Citable do
cuments']]
    top15['pop'] = top15.apply(lambda row: row['Energy Supply']/row['Energy Supp
ly per Capita'], axis = 1)
    top15['docpercap'] = top15.apply(lambda row: row['Citable documents']/row['p
op'], axis = 1)
    corr, pval = stats.pearsonr(top15['Energy Supply per Capita'], top15['docper
cap'])
    return corr
    raise NotImplementedError()
answer_nine()
```

Out[20]:

```
0.7940010435442942
```

In [21]:

```
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['PopEs
t']
    Top15.plot(x='Citable docs per Capita', y='Energy Supply per Capita', kind=
'scatter', xlim=[0, 0.0006])
```

In [22]:

```
#plot9()
```

In [23]:

```
assert answer_nine() >= -1. and answer_nine() <= 1., "Q9: A valid correlation sh
ould between -1 to 1!"
```

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.*

In [24]:

```
def answer_ten():
    # YOUR CODE HERE
    top15 = answer_one()['% Renewable'].to_frame()
    med = top15['% Renewable'].median()
    #Get country at median
    top15['HighRenew'] = [1 if x >= med else 0 for x in top15['% Renewable']]
    return top15['HighRenew']
    raise NotImplementedError()
answer_ten()
```

Out[24]:

```
Country
China                 1
United States         0
Japan                 0
United Kingdom        0
Russian Federation    1
Canada                1
Germany               1
India                 0
France                1
South Korea           0
Italy                 1
Spain                 1
Iran                  0
Australia             0
Brazil                1
Name: HighRenew, dtype: int64
```

In [25]:

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

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']`

In [64]:

```
def answer_eleven():
    # YOUR CODE HERE
    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 = answer_one()[['Energy Supply','Energy Supply per Capita']].reset_ind
ex()
    top15['pop'] = top15.apply(lambda row: row['Energy Supply']/row['Energy Supp
ly per Capita'], axis = 1)
    top15['Continent'] = top15.apply(lambda row: ContinentDict[row['Country']],a
xis = 1)
    result = top15.set_index('Continent').groupby(level = 0)['pop'].agg(
        {'size':np.size, 'sum': np.sum, 'mean': np.nanmean, 'std' : np.nanstd})
    result = result[['size', 'sum', 'mean', 'std']]
    #Continent_list = top15['Continent'].unique()

    return result
    raise NotImplementedError()
answer_eleven()
```

Out[64]:

	size	sum	mean	std
Continent				
Asia	5.0	2.898666e+09	5.797333e+08	6.790979e+08
Australia	1.0	2.331602e+07	2.331602e+07	NaN
Europe	6.0	4.579297e+08	7.632161e+07	3.464767e+07
North America	2.0	3.528552e+08	1.764276e+08	1.996696e+08
South America	1.0	2.059153e+08	2.059153e+08	NaN

In [27]:

```
assert type(answer_eleven()) == pd.DataFrame, "Q11: You should return a DataFram
e!"

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

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

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.*

In [67]:

```
def answer_twelve():
    # YOUR CODE HERE
    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 = answer_one()
    Top15 = Top15.reset_index()
    Top15['Continent'] =Top15.apply(lambda row: ContinentDict[row['Country']],ax
is = 1)
    Top15['% Renewable'] = pd.cut(Top15['% Renewable'],5)
    result = Top15.groupby(['Continent', '% Renewable']).size()
    return result
    raise NotImplementedError()
answer_twelve()
```

Out[67]:

```
Continent      % Renewable
Asia           (2.212, 15.753]     4
               (15.753, 29.227]    1
Australia      (2.212, 15.753]     1
Europe         (2.212, 15.753]     1
               (15.753, 29.227]    3
               (29.227, 42.701]    2
North America  (2.212, 15.753]     1
               (56.174, 69.648]    1
South America  (56.174, 69.648]    1
dtype: int64
```

In [66]:

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

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

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*

In [30]:

```
def convstr(num):
    import string as s
    string = str(num)

    fullstopindex = string.find(".")
    i = fullstopindex
    cycle = 0
    while i > 0:
        i = i - 1
        cycle+=1
        if (i == 0):
            pass
        elif (cycle %3 == 0):
            string = string[:i] + ',' + string[i:]
            # Length of string increases.
        else:
            pass
    return string
def answer_thirteen():
    top15 = answer_one()[['Energy Supply','Energy Supply per Capita']]
    top15['PopEst'] = top15.apply(lambda row: row['Energy Supply']/row['Energy S
upply per Capita'], axis = 1)
    top15['PopEst'] = top15.apply(lambda row: convstr(row['PopEst']), axis = 1)
    return top15['PopEst']
    raise NotImplementedError()
answer_thirteen()
```

Out[30]:

```
Country
China                     1,367,645,161.2903225
United States              317,615,384.61538464
Japan                      127,409,395.97315437
United Kingdom             63,870,967.741935484
Russian Federation                143,500,000.0
Canada                      35,239,864.86486486
Germany                     80,369,696.96969697
India                     1,276,730,769.2307692
France                      63,837,349.39759036
South Korea                49,805,429.864253394
Italy                      59,908,256.880733944
Spain                        46,443,396.2264151
Iran                        77,075,630.25210084
Australia                  23,316,017.316017315
Brazil                     205,915,254.23728815
Name: PopEst, dtype: object
```

In [31]:

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

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

Optional

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

In [32]:

```
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','#377eb
8','#4daf4a','#e41a1c',
                       '#4daf4a','#e41a1c','#4daf4a','#4daf4a','#e41a1c','#dede0
0','#ff7f00'],
                    xticks=range(1,16), s=6*Top15['2014']/10**10, alpha=.75, fig
size=[16,6]);

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

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

In [34]:

```
#plot_optional()
```

In [ ]:

```

```