*You are currently looking at **version 1.2** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ (https://www.coursera.org/learn/python-data-analysis/resources/0dhYG) course resource.*

Assignment 2 - Pandas Introduction

All questions are weighted the same in this assignment.

Part 1

The following code loads the olympics dataset (olympics.csv), which was derrived from the Wikipedia entry on All Time Olympic Games Medals (https://en.wikipedia.org/wiki/All-time_Olympic_Games_medal_table), and does some basic data cleaning.

The columns are organized as # of Summer games, Summer medals, # of Winter games, Winter medals, total # number of games, total # of medals. Use this dataset to answer the questions below.

In [72]:

```
import pandas as pd

df = pd.read_csv('olympics.csv', index_col=0, skiprows=1)

for col in df.columns:
    if col[:2]=='01':
        df.rename(columns={col:'Gold'+col[4:]}, inplace=True)
    if col[:2]=='02':
        df.rename(columns={col:'Silver'+col[4:]}, inplace=True)
    if col[:2]=='03':
        df.rename(columns={col:'Bronze'+col[4:]}, inplace=True)
    if col[:1]=='№':
        df.rename(columns={col:'#'+col[1:]}, inplace=True)

# print(df.index)
"""
Index(['Afghanistan (AFG)', 'Algeria (ALG)', 'Argentina (ARG)',
       'Armenia (ARM)', 'Australasia (ANZ) [ANZ]', 'Australia (AUS) [AUS] [Z]',
"""

names_ids = df.index.str.split('\s\(') # split the index by '('
#  print(names_ids)
"""
Index([                                  ['Afghanistan', 'AFG)'],
                                             ['Algeria', 'ALG)'],

       ...

                             ['Mixed team', 'ZZX) [ZZX]'],
                                               ['Totals']],
      dtype='object', length=147)
"""

df.index = names_ids.str[0] # the [0] element is the country name (new index)
# print(df.index)
"""
Index(['Afghanistan', 'Algeria', 'Argentina', 'Armenia', 'Australasia',
       'Australia', 'Austria', 'Azerbaijan', 'Bahamas', 'Bahrain',
       ...
       'Uzbekistan', 'Venezuela', 'Vietnam', 'Virgin Islands', 'Yugoslavia',
       'Independent Olympic Participants', 'Zambia', 'Zimbabwe', 'Mixed team',
       'Totals'],
      dtype='object', length=147)
"""

df['ID'] = names_ids.str[1].str[:3] # the [1] element is the abbreviation or ID (take first 3 characters from that)
# print(df['ID'])
"""
Afghanistan                         AFG
Algeria                             ALG
                                   ...
Independent Olympic Participants    IOP
Totals                              NaN
Name: ID, Length: 147, dtype: object
"""
```

```
df = df.drop('Totals')
df.head()
```

Out[72]:

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	# Games	Gold.2	Silver.2	Bronze.2	Combined total	ID
Afghanistan	13	0	0	2	2	0	0	0	0	0	13	0	0	2	2	AFG
Algeria	12	5	2	8	15	3	0	0	0	0	15	5	2	8	15	ALG
Argentina	23	18	24	28	70	18	0	0	0	0	41	18	24	28	70	ARG
Armenia	5	1	2	9	12	6	0	0	0	0	11	1	2	9	12	ARM
Australasia	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12	ANZ

Question 0 (Example)

What is the first country in df?

This function should return a Series.

In [73]:

```
# You should write your whole answer within the function provided. The autograder will call
# this function and compare the return value against the correct solution value
def answer_zero():
    # This function returns the row for Afghanistan, which is a Series object. The assignment
    # question description will tell you the general format the autograder is expecting
    return df.iloc[0]

# You can examine what your function returns by calling it in the cell. If you have questions
# about the assignment formats, check out the discussion forums for any FAQs
answer_zero()
```

Out[73]:

```
# Summer           13
Gold                0
Silver              0
Bronze              2
Total               2
# Winter            0
Gold.1              0
Silver.1            0
Bronze.1            0
Total.1             0
# Games            13
Gold.2              0
Silver.2            0
Bronze.2            2
Combined total      2
ID                AFG
Name: Afghanistan, dtype: object
```

### Question 1 Which country has won the most gold medals in summer games? *This function should return a single string value.*

In [74]:

```
def answer_one():
    max_gold = max(df['Gold'])
    return df[df['Gold'] == max_gold].index[0]

answer_one()
```

Out[74]:

```
'United States'
```

Question 2

Which country had the biggest difference between their summer and winter gold medal counts?

This function should return a single string value.

In [75]:

```
def answer_two():
    max_diff = max(df['Gold'] - df['Gold.1'])
    return df[(df['Gold'] - df['Gold.1']) == max_diff].index[0]

answer_two()
```

Out[75]:

```
'United States'
```

In [76]:

```
def answer_two():
    copy_df = df.copy()
    copy_df['Diff'] = copy_df['Gold'] - copy_df['Gold.1']
    # print(copy_df['Diff'])
    most_diff = max(copy_df['Diff'])
    return (str(copy_df[copy_df['Diff'] == most_diff].index[0]))

answer_two()
```

Out[76]:

```
'United States'
```

Question 3

Which country has the biggest difference between their summer gold medal counts and winter gold medal counts relative to their total gold medal count?

$$\frac{Summer\ Gold - Winter\ Gold}{Total\ Gold}$$

Only include countries that have won at least 1 gold in both summer and winter.

This function should return a single string value.

In [77]:

```
def answer_three():
    copy = df[df['Gold'] > 0]
    copy = df[df['Gold.1'] > 0]
    copy['Radio'] = (df['Gold'] - df['Gold.1']) / (df['Gold'] + df['Gold.1'])
    max_radio = max(copy['Radio'])
    return copy[copy['Radio'] == max_radio].index[0]
answer_three()
```

```
C:\Users\asus\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarning:
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: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

Out[77]:

```
'Bulgaria'
```

In [78]:

```
import numpy as np
def answer_three():
    copy_df = df.copy()
    copy_df = copy_df.where(df['Gold'] > 0)
    copy_df = copy_df.where(df['Gold.1'] > 0)
    copy_df['Diff'] = copy_df['Gold'] - copy_df['Gold.1']
    copy_df['Gold_Ratio'] = copy_df['Diff'] / (copy_df['Gold'] + copy_df['Gold.1'])
    copy_df_final = copy_df[np.isfinite(copy_df['Gold_Ratio'])]
    max_ratio = max(copy_df_final['Gold_Ratio'])

    return (str(copy_df_final[copy_df_final['Gold_Ratio'] == max_ratio].index[0]))
answer_three()
```

Out[78]:

```
'Bulgaria'
```

Question 4

Write a function that creates a Series called "Points" which is a weighted value where each gold medal (`Gold.2`) counts for 3 points, silver medals (`Silver.2`) for 2 points, and bronze medals (`Bronze.2`) for 1 point. The function should return only the column (a Series object) which you created, with the country names as indices.

This function should return a Series named `Points` *of length 146*

In [79]:

```
def answer_four():
    df['Points'] = df['Gold.2'] * 3 + df['Silver.2'] * 2 + df['Bronze.2']
    # print(len(df['Points']))
    return pd.Series(df['Points'])

answer_four()
```

Out[79]:

```
Afghanistan                        2
Algeria                           27
Argentina                        130
Armenia                           16
Australasia                       22
                                ... 
Yugoslavia                       171
Independent Olympic Participants   4
Zambia                             3
Zimbabwe                          18
Mixed team                        38
Name: Points, Length: 146, dtype: int64
```

Part 2

For the next set of questions, we will be using census data from the United States Census Bureau (http://www.census.gov). Counties are political and geographic subdivisions of states in the United States. This dataset contains population data for counties and states in the US from 2010 to 2015. See this document (https://www2.census.gov/programs-surveys/popest/technical-documentation/file-layouts/2010-2015/co-est2015-alldata.pdf) for a description of the variable names.

The census dataset (census.csv) should be loaded as census_df. Answer questions using this as appropriate.

Question 5

Which state has the most counties in it? (hint: consider the sumlevel key carefully! You'll need this for future questions too...)

This function should return a single string value.

In [80]:

```
import pandas as pd
census_df = pd.read_csv('census.csv')
census_df
```

Out[80]:

	SUMLEV	REGION	DIVISION	STATE	COUNTY	STNAME	CTYNAME	CENSUS2010POP	ESTIMATESBASE2010	POPESTIMATE2010	...	RDOMESTICMIG2011	RDOMESTICMIG2012	RDOMESTICMIG2013	RDOMESTICMIG2014	RDO
0	40	3	6	1	0	Alabama	Alabama	4779736	4780127	4785161	...	0.002295	-0.193196	0.381066	0.582002	
1	50	3	6	1	1	Alabama	Autauga County	54571	54571	54660	...	7.242091	-2.915927	-3.012349	2.265971	
2	50	3	6	1	3	Alabama	Baldwin County	182265	182265	183193	...	14.832960	17.647293	21.845705	19.243287	
3	50	3	6	1	5	Alabama	Barbour County	27457	27457	27341	...	-4.728132	-2.500690	-7.056824	-3.904217	
4	50	3	6	1	7	Alabama	Bibb County	22915	22919	22861	...	-5.527043	-5.068871	-6.201001	-0.177537	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3188	50	4	8	56	37	Wyoming	Sweetwater County	43806	43806	43593	...	1.072643	16.243199	-5.339774	-14.252889	
3189	50	4	8	56	39	Wyoming	Teton County	21294	21294	21297	...	-1.589565	0.972695	19.525929	14.143021	
3190	50	4	8	56	41	Wyoming	Uinta County	21118	21118	21102	...	-17.755986	-4.916350	-6.902954	-14.215862	
3191	50	4	8	56	43	Wyoming	Washakie County	8533	8533	8545	...	-11.637475	-0.827815	-2.013502	-17.781491	
3192	50	4	8	56	45	Wyoming	Weston County	7208	7208	7181	...	-11.752361	-8.040059	12.372583	1.533635	

3193 rows × 100 columns

In [81]:

```
def answer_five():
    df = census_df[census_df['SUMLEV'] == 50]
    return df['STNAME'].value_counts().index[0]

answer_five()
```

Out[81]:

```
'Texas'
```

In [82]:

```
def answer_five():
    counties_df = census_df[census_df['SUMLEV'] == 50]
    x = counties_df.groupby('STNAME').count()['CTYNAME']
    ans = x.idxmax()
    return ans

answer_five()
```

Out[82]:

```
'Texas'
```

Question 6

Only looking at the three most populous counties for each state, what are the three most populous states (in order of highest population to lowest population)? Use `CENSUS2010POP`.

This function should return a list of string values.

In [83]:

```
def answer_six():
    df = census_df[census_df['SUMLEV'] == 50]
    df['sort_result'] = df['CENSUS2010POP'].groupby(df['STNAME']).rank(ascending = False)
    max_top3 = df[df['sort_result']<=3]
    ans = max_top3.groupby(max_top3['STNAME'])['CENSUS2010POP'].sum().sort_values(ascending = False).head(3).index.tolist()
    return ans
answer_six()
```

```
C:\Users\asus\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
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: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[83]:

```
['California', 'Texas', 'Illinois']
```

In [84]:

```
def answer_six():

    x = census_df[census_df['SUMLEV'] == 50]
    x['sort_result'] = x['CENSUS2010POP'].groupby(x['STNAME']).rank(ascending = False)
    max_3 = x[x['sort_result']<=3]
    summed_max_3 = max_3.groupby(max_3['STNAME'])['CENSUS2010POP'].sum().sort_values(ascending = False)

    return list(summed_max_3.index[:3].values)
answer_six()
```

```
C:\Users\asus\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarning:
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: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

Out[84]:

```
['California', 'Texas', 'Illinois']
```

Question 7

Which county has had the largest absolute change in population within the period 2010-2015? (Hint: population values are stored in columns POPESTIMATE2010 through POPESTIMATE2015, you need to consider all six columns.)

e.g. If County Population in the 5 year period is 100, 120, 80, 105, 100, 130, then its largest change in the period would be |130-80| = 50.

This function should return a single string value.

In [85]:

```python
def answer_seven():
    counties_df = census_df[census_df['SUMLEV'] == 50]
    counties_df = counties_df.set_index('CTYNAME')
    columns = ['POPESTIMATE2010',
               'POPESTIMATE2011',
               'POPESTIMATE2012',
               'POPESTIMATE2013',
               'POPESTIMATE2014',
               'POPESTIMATE2015']
    # calculate the max et the min value of every horizontal axis (axis = 0)
    # vertical axis: axis = 1
    max_pop = counties_df[columns].max(axis=1)
    min_pop = counties_df[columns].min(axis=1)
    counties_df['diff'] = max_pop - min_pop
    return counties_df['diff'].idxmax()

answer_seven()
```

Out[85]:

```
'Harris County'
```

Question 8

In this datafile, the United States is broken up into four regions using the "REGION" column.

Create a query that finds the counties that belong to regions 1 or 2, whose name starts with 'Washington', and whose POPESTIMATE2015 was greater than their POPESTIMATE 2014.

This function should return a 5x2 DataFrame with the columns = ['STNAME', 'CTYNAME'] and the same index ID as the census_df (sorted ascending by index).

In [86]:

```python
def answer_eight():
    counties_df = census_df[census_df['SUMLEV'] == 50]
    ans = counties_df[((counties_df['REGION'] == 1) | (counties_df['REGION'] == 2)) & (counties_df['CTYNAME'] == 'Washington County') & (counties_df['POPESTIMATE2015'] > counties_df['POPESTIMATE2014'])][['STNAME','CTYNAME']]
    return ans

answer_eight()
```

Out[86]:

	STNAME	CTYNAME
896	Iowa	Washington County
1419	Minnesota	Washington County
2345	Pennsylvania	Washington County
2355	Rhode Island	Washington County
3163	Wisconsin	Washington County

In [ ]:

*You are currently looking at **version 1.5** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ (https://www.coursera.org/learn/python-data-analysis/resources/0dhYG) course resource.*

Assignment 3 - More Pandas

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. And of course, the discussion forums are open for interaction with your peers and the course staff.

Question 1 (20%)

Load the energy data from the file `Energy Indicators.xls`, which is a list of indicators of energy supply and renewable electricity production (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 (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 `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 `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.

In [2]:

```
import pandas as pd
import numpy as np

def answer_one():
    # skipfooter: Rows at the end to skip (0-indexed)
    energy = pd.read_excel('Energy Indicators.xls', skiprows=17, skipfooter=38)

    # get rid of the 2 first columns
    cols = ['Unnamed: 2', 'Petajoules', 'Gigajoules', '%']
    energy = energy[cols]
    energy.columns = ['Country', 'Energy Supply', 'Energy Supply per Capita', '% Renewable']

    # For all countries which have missing data (e.g. data with "...")
    # make sure this is reflected as np.NaN values.
    energy = energy.replace('...', np.nan)


    # Convert Energy Supply to gigajoules (there are 1,000,000 gigajoules in a petajoule)
    energy['Energy Supply'] = energy['Energy Supply'] * 1000000

    # Remove the numbers in the country name
    energy['Country'] = energy['Country'].str.replace(r"[0-9]","")

    energy['Country'] = energy['Country'].replace({
        'China, Hong Kong Special Administrative Region':'Hong Kong',
        'United Kingdom of Great Britain and Northern Ireland':'United Kingdom',
        'Republic of Korea':'South Korea',
        'United States of America':'United States',
        'Iran (Islamic Republic of)':'Iran',
        'Bolivia (Plurinational State of)':'Bolivia'})

    # This removed all instances of where there were parentheses with words in them
    energy['Country'] = energy['Country'].str.replace(r" \(.*\)","")

    GDP = pd.read_csv("world_bank.csv", skiprows=4)
    GDP['Country Name'] = GDP['Country Name'].replace({'Korea, Rep.' : 'South Korea',
                                                       'Iran, Islamic Rep.' : 'Iran',
                                                       'Hong Kong SAR, China' : 'Hong Kong'})

    ScimEn = pd.read_excel('scimagojr-3.xlsx')

    # 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).
    cols_GDP = ['Country Name','2006','2007','2008','2009','2010','2011','2012','2013','2014','2015']
    GDP_merge = GDP[cols_GDP]
    GDP_merge.columns = ['Country','2006','2007','2008','2009','2010','2011','2012','2013','2014','2015']

    ScimEn_merge = ScimEn[:15]

    df0 = pd.merge(ScimEn_merge, energy, how='inner', left_on='Country', right_on='Country')
    df = pd.merge(df0, GDP_merge, how='inner', left_on='Country', right_on='Country')
```

```python
    # 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'].
    df = df.set_index('Country')
    columns = ['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']
    df = df[columns]
    return df

answer_one()
```

Out[2]:

	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
Country																		
China	1	127050	126767	597237	411683	4.70	138	1.271910e+11	93.0	19.754910	3.992331e+12	4.559041e+12	4.997775e+12	5.459247e+12	6.039659e+12	6.612490e+12	7.124978e+12	7.672448e+12
United States	2	96661	94747	792274	265436	8.20	230	9.083800e+10	286.0	11.570980	1.479230e+13	1.505540e+13	1.501149e+13	1.459484e+13	1.496437e+13	1.520402e+13	1.554216e+13	1.577367e+13
Japan	3	30504	30287	223024	61554	7.31	134	1.898400e+10	149.0	10.232820	5.496542e+12	5.617036e+12	5.558527e+12	5.251308e+12	5.498718e+12	5.473738e+12	5.569102e+12	5.644659e+12
United Kingdom	4	20944	20357	206091	37874	9.84	139	7.920000e+09	124.0	10.600470	2.419631e+12	2.482203e+12	2.470614e+12	2.367048e+12	2.403504e+12	2.450911e+12	2.479809e+12	2.533370e+12
Russian Federation	5	18534	18301	34266	12422	1.85	57	3.070900e+10	214.0	17.288680	1.385793e+12	1.504071e+12	1.583004e+12	1.459199e+12	1.524917e+12	1.589943e+12	1.645876e+12	1.666934e+12
Canada	6	17899	17620	215003	40930	12.01	149	1.043100e+10	296.0	61.945430	1.564469e+12	1.596740e+12	1.612713e+12	1.565145e+12	1.613406e+12	1.664087e+12	1.693133e+12	1.730688e+12
Germany	7	17027	16831	140566	27426	8.26	126	1.326100e+10	165.0	17.901530	3.332891e+12	3.441561e+12	3.478809e+12	3.283340e+12	3.417298e+12	3.542371e+12	3.556724e+12	3.567317e+12
India	8	15005	14841	128763	37209	8.58	115	3.319500e+10	26.0	14.969080	1.265894e+12	1.374865e+12	1.428361e+12	1.549483e+12	1.708459e+12	1.821872e+12	1.924235e+12	2.051982e+12
France	9	13153	12973	130632	28601	9.93	114	1.059700e+10	166.0	17.020280	2.607840e+12	2.669424e+12	2.674637e+12	2.595967e+12	2.646995e+12	2.702032e+12	2.706968e+12	2.722567e+12
South Korea	10	11983	11923	114675	22595	9.57	104	1.100700e+10	221.0	2.279353	9.410199e+11	9.924316e+11	1.020510e+12	1.027730e+12	1.094499e+12	1.134796e+12	1.160809e+12	1.194429e+12
Italy	11	10964	10794	111850	26661	10.20	106	6.530000e+09	109.0	33.667230	2.202170e+12	2.234627e+12	2.211154e+12	2.089938e+12	2.125185e+12	2.137439e+12	2.077184e+12	2.040871e+12
Spain	12	9428	9330	123336	23964	13.08	115	4.923000e+09	106.0	37.968590	1.414823e+12	1.468146e+12	1.484530e+12	1.431475e+12	1.431673e+12	1.417355e+12	1.380216e+12	1.357139e+12
Iran	13	8896	8819	57470	19125	6.46	72	9.172000e+09	119.0	5.707721	3.895523e+11	4.250646e+11	4.289909e+11	4.389208e+11	4.677902e+11	4.853309e+11	4.532569e+11	4.445926e+11
Australia	14	8831	8725	90765	15606	10.28	107	5.386000e+09	231.0	11.810810	1.021939e+12	1.060340e+12	1.099644e+12	1.119654e+12	1.142251e+12	1.169431e+12	1.211913e+12	1.241484e+12
Brazil	15	8668	8596	60702	14396	7.00	86	1.214900e+10	59.0	69.648030	1.845080e+12	1.957118e+12	2.056809e+12	2.054215e+12	2.208872e+12	2.295245e+12	2.339209e+12	2.409740e+12

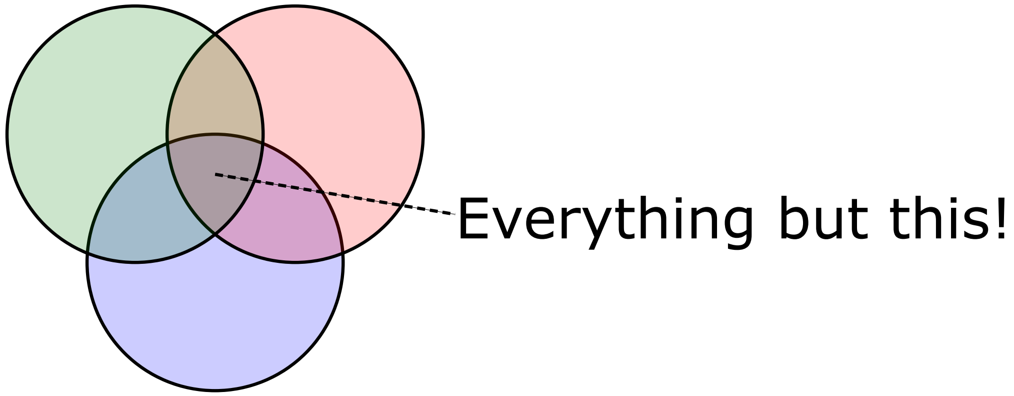
Question 2 (6.6%)

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 [3]:

```
%%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>
```



In [4]:

```
def answer_two():
    # skipfooter: Rows at the end to skip (0-indexed)
    energy = pd.read_excel('Energy Indicators.xls', skiprows=17, skipfooter=38)

    # get rid of the 2 first columns
    cols = ['Unnamed: 2', 'Petajoules', 'Gigajoules', '%']
    energy = energy[cols]
    energy.columns = ['Country', 'Energy Supply', 'Energy Supply per Capita', '% Renewable']

    # For all countries which have missing data (e.g. data with "...")
    # make sure this is reflected as np.NaN values.
    energy = energy.replace('...', np.nan)


    # Convert Energy Supply to gigajoules (there are 1,000,000 gigajoules in a petajoule)
    energy['Energy Supply'] = energy['Energy Supply'] * 1000000

    energy['Country'] = energy['Country'].str.replace(r"[0-9]","")

    energy['Country'] = energy['Country'].replace({
        'China, Hong Kong Special Administrative Region':'Hong Kong',
        'United Kingdom of Great Britain and Northern Ireland':'United Kingdom',
        'Republic of Korea':'South Korea',
        'United States of America':'United States',
        'Iran (Islamic Republic of)':'Iran',
        'Bolivia (Plurinational State of)':'Bolivia'})

    # This removed all instances of where there were parentheses with words in them
    energy['Country'] = energy['Country'].str.replace(r" \(.*\)","")

    GDP = pd.read_csv("world_bank.csv", skiprows=4)
    GDP['Country Name'] = GDP['Country Name'].replace({'Korea, Rep.' : 'South Korea',
                                                      'Iran, Islamic Rep.' : 'Iran',
                                                      'Hong Kong SAR, China' : 'Hong Kong'})

    ScimEn = pd.read_excel('scimagojr-3.xlsx')

    df_outer0 = pd.merge(ScimEn, energy, how='outer', left_on='Country', right_on='Country')
    df_outer = pd.merge(df_outer0, GDP, how='outer', left_on='Country', right_on='Country Name')
    len_outer = len(df_outer)
    # print(len_outer)

    df_inner0 = pd.merge(ScimEn, energy, how='inner', left_on='Country', right_on='Country')
    df_inner = pd.merge(df_inner0, GDP, how='inner', left_on='Country', right_on='Country Name')
    len_inner = len(df_inner)
    # print(len_inner)

    return (len_outer)-(len_inner)

answer_two()
```

Out[4]:

```
156
```

Answer the following questions in the context of only the top 15 countries by Scimagojr Rank (aka the DataFrame returned by `answer_one()`)

Question 3 (6.6%)

What is the average GDP over the last 10 years for each country? (exclude missing values from this calculation.)

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

In [5]:

```
import numpy as np
def mean_top15(row):
    data = row[['2006','2007','2008','2009','2010','2011','2012','2013','2014','2015']]
    return pd.Series({'mean': np.mean(data)})

def answer_three():
    Top15 = answer_one()
    avgGDP_notOrdered = Top15.apply(mean_top15, axis=1)
    avgGDP = avgGDP_notOrdered.sort_values(by='mean', ascending = False)
    return avgGDP

answer_three()
```

Out[5]:

	mean
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

In [6]:

```
def answer_three_alter():
    import numpy as np
    Top15 = answer_one()
    columns = ['2006','2007','2008','2009','2010','2011','2012','2013','2014','2015']
    Top15['Mean'] = Top15[columns].mean(axis=1)
    avgGDP = Top15.sort_values(by = 'Mean', ascending = False)['Mean']

    return avgGDP
answer_three_alter()
```

Out[6]:

```
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
```

Question 4 (6.6%)

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 [51]:

```
def answer_four():
    Top15 = answer_one()

    avgGDP = answer_three()
    Top6th_Country = avgGDP.index[5]

    Top6th = Top15.loc[Top6th_Country]

    """
    Or:
    Top15 = Top15.reset_index()
    Top6th = Top15[Top15['Country'] == Top6th_Country]
    span = (Top6th['2015'] - Top6th['2006']).value[0]
    """

    span = Top6th['2015'] - Top6th['2006']
    return span
answer_four()
```

Out[51]:

```
246702696075.3999
```

In [52]:

```
def answer_four_alter():
    import pandas as pd
    import numpy as np
    Top15 = answer_one()
    columns = ['2006','2007','2008','2009','2010','2011','2012','2013','2014','2015']
    Top15['Mean'] = Top15[columns].mean(axis=1)
    avgGDP = Top15.sort_values(by = 'Mean', ascending = False)['Mean']
    target = avgGDP.index[5]

    target_data = Top15.loc[target]
    ans = target_data['2015'] - target_data['2006']

    return ans
answer_four_alter()
```

Out[52]:

```
246702696075.3999
```

Question 5 (6.6%)

What is the mean `Energy Supply per Capita` ?

This function should return a single number.

In [59]:

```
def answer_five():
    Top15 = answer_one()

    return Top15['Energy Supply per Capita'].mean(axis=0)

answer_five()
```

Out[59]:

```
157.6
```

Question 6 (6.6%)

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 [70]:

```
def answer_six():
    Top15 = answer_one()
    max_renewable = Top15['% Renewable'].max()
    country = Top15[Top15['% Renewable'] == max_renewable].index[0]
    # country = Top15[Top15['% Renewable'] == max_renewable].index
    # print(country)
    # Index(['Brazil'], dtype='object', name='Country')
    return country, max_renewable

answer_six()
```

Out[70]:

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

Question 7 (6.6%)

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 [72]:

```
def answer_seven():
    Top15 = answer_one()
    Top15['Ratio_Citations'] = Top15['Self-citations'] / Top15['Citations']
    max_ratio = Top15['Ratio_Citations'].max()
    country = Top15[Top15['Ratio_Citations'] == max_ratio].index[0]
    return (country, max_ratio)

answer_seven()
```

Out[72]:

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

Question 8 (6.6%)

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 a single string value.

```
In [81]: def answer_eight():
             Top15 = answer_one()
             Top15['Estimated_Population'] = Top15['Energy Supply'] / Top15['Energy Supply per Capita']
             population = Top15.sort_values(by='Estimated_Population', ascending=False)['Estimated_Population']
             third_population = Top15[Top15['Estimated_Population'] == population.iloc[2]].index[0]
             return third_population

         answer_eight()
```

```
Out[81]: 'United States'
```

```
In [83]: def answer_eight_alter():
             Top15 = answer_one()
             columns = ['Energy Supply','Energy Supply per Capita']
             target = Top15[columns]
             target['Population'] = Top15['Energy Supply'] / Top15['Energy Supply per Capita']

             ans = target.sort_values(by = 'Population', ascending = False).iloc[2].name

             return ans
         answer_eight_alter()
```

```
C:\Users\asus\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: SettingWithCopyWarning:
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: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

```
Out[83]: 'United States'
```

Question 9 (6.6%)

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.

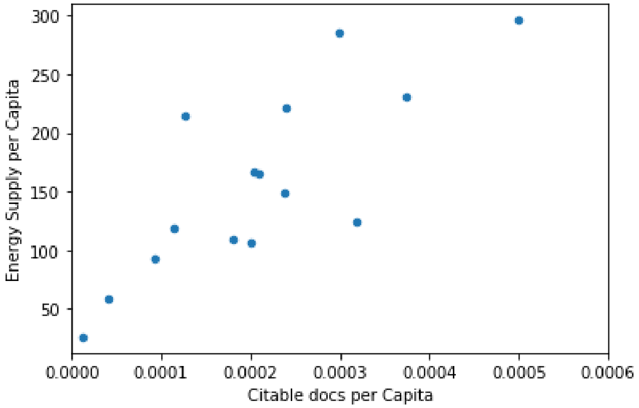
(Optional: Use the built-in function `plot9()` to visualize the relationship between Energy Supply per Capita vs. Citable docs per Capita)

```
In [88]: def answer_nine():
             Top15 = answer_one()
             Top15['Estimated_Population'] = Top15['Energy Supply'] / Top15['Energy Supply per Capita']
             Top15['Doc per Person'] = Top15['Citable documents'] / Top15['Estimated_Population']
             # Top15['Corr_Citation_Energy'] = Top15['Energy Supply per Capita'].corr(Top15['Doc per Person'])

             return Top15['Doc per Person'].corr(Top15['Energy Supply per Capita'])
         answer_nine()
```

```
Out[88]: 0.7940010435442946
```

```
In [89]: 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])
```

```
In [90]: plot9() # Be sure to comment out plot9() before submitting the assignment!
```



Question 10 (6.6%)

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 [117]:

```
import numpy as np
def isAboveMedian(row):
    Top15 = answer_one()
    median = np.nanmedian(Top15['% Renewable'])
    data = row['% Renewable']
    row['HighRenew'] = 1 if data >= median else 0
    return pd.Series(row['HighRenew'])

def answer_ten():
    Top15 = answer_one()
    return Top15.apply(isAboveMedian, axis=1).sort_index()

answer_ten()
```

Out[117]:

```
                      0
           Country
         Australia  0.0
            Brazil  1.0
            Canada  1.0
             China  1.0
            France  1.0
           Germany  1.0
             India  0.0
              Iran  0.0
             Italy  1.0
             Japan  0.0
Russian Federation  1.0
       South Korea  0.0
             Spain  1.0
    United Kingdom  0.0
     United States  0.0
```

In [107]:

```
def answer_ten_alter():
    import pandas as pd
    Top15 = answer_one()
    med = Top15['% Renewable'].median()
    Top15['HighRenew'] = [1 if x >= med else 0 for x in Top15['% Renewable']]
    ans = Top15['HighRenew']
    return pd.Series(ans).sort_index()
answer_ten_alter()
```

Out[107]:

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

Question 11 (6.6%)

Use the following dictionary to group the Countries by Continent, then create a dateframe 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 [146]:

```python
def answer_eleven():
    Top15 = answer_one()
    Top15 = Top15.reset_index()
    Top15['Estimated Population'] = Top15['Energy Supply'] / Top15['Energy Supply per Capita']

    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['Continent'] = [ContinentDict[country] for country in Top15['Country']]
    Top15 = Top15.set_index('Continent')
    summary = Top15.groupby(level=0)['Estimated Population'].agg({'sample size': np.size,
                                                           'sum': np.sum,
                                                           'average': np.nanmean,
                                                           'standard deviation' : np.nanstd})

    return summary

answer_eleven()
```

```
C:\Users\asus\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: FutureWarning: using a dict on a Series for aggregation
is deprecated and will be removed in a future version. Use                 named aggregation instead.

    >>> grouper.agg(name_1=func_1, name_2=func_2)
```

Out[146]:

	sample size	sum	average	standard deviation
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 [147]:

```
def answer_eleven_alter():
    import pandas as pd
    import numpy as np
    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['PopEst'] = (Top15['Energy Supply'] / Top15['Energy Supply per Capita'])

    Top15 = Top15.reset_index()
    Top15['Continent'] = [ContinentDict[country] for country in Top15['Country']]
#     print(Top15['Continent'])
#     print(ContinentDict.values())
#     Top15['Continent'] = [ContinentDict[country] for country in Top15['Country']]

    target = Top15.set_index('Continent').groupby(level = 0)['PopEst'].agg({'size':np.size,
                                                                            'sum':np.sum,
                                                                            'mean':np.mean,
                                                                            'std':np.std})
    ans = target[['size', 'sum', 'mean', 'std']]
    return ans

answer_eleven_alter()
```

```
C:\Users\asus\Anaconda3\lib\site-packages\ipykernel_launcher.py:33: FutureWarning: using a dict on a Series for aggregation
is deprecated and will be removed in a future version. Use                 named aggregation instead.

    >>> grouper.agg(name_1=func_1, name_2=func_2)
```

Out[147]:

	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

Question 12 (6.6%)

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 [154]:

```
import pandas as pd
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 = Top15.reset_index()
    Top15['Continent'] = [ContinentDict[country] for country in Top15['Country']]
    Top15['bins'] = pd.cut(Top15['% Renewable'], 5)
    Top15 = Top15.groupby(['Continent', 'bins'])
    return Top15.size()

answer_twelve()
```

Out[154]:

```
Continent      bins
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
```

Question 13 (6.6%)

Convert the Population Estimate series to a string with thousands separator (using commas). Do not round the results.

e.g. 317615384.61538464 -> 317,615,384.61538464

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

In [162]:

```
import pandas as pd
def answer_thirteen():
    Top15 = answer_one()
    Top15['PopEst'] = Top15['Energy Supply'] / Top15['Energy Supply per Capita']
    Top15['PopEst'] = Top15['PopEst'].apply(lambda x: "{:,}".format(x))
    return pd.Series(Top15['PopEst'])

answer_thirteen()
```

Out[162]:

```
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
```

Optional

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

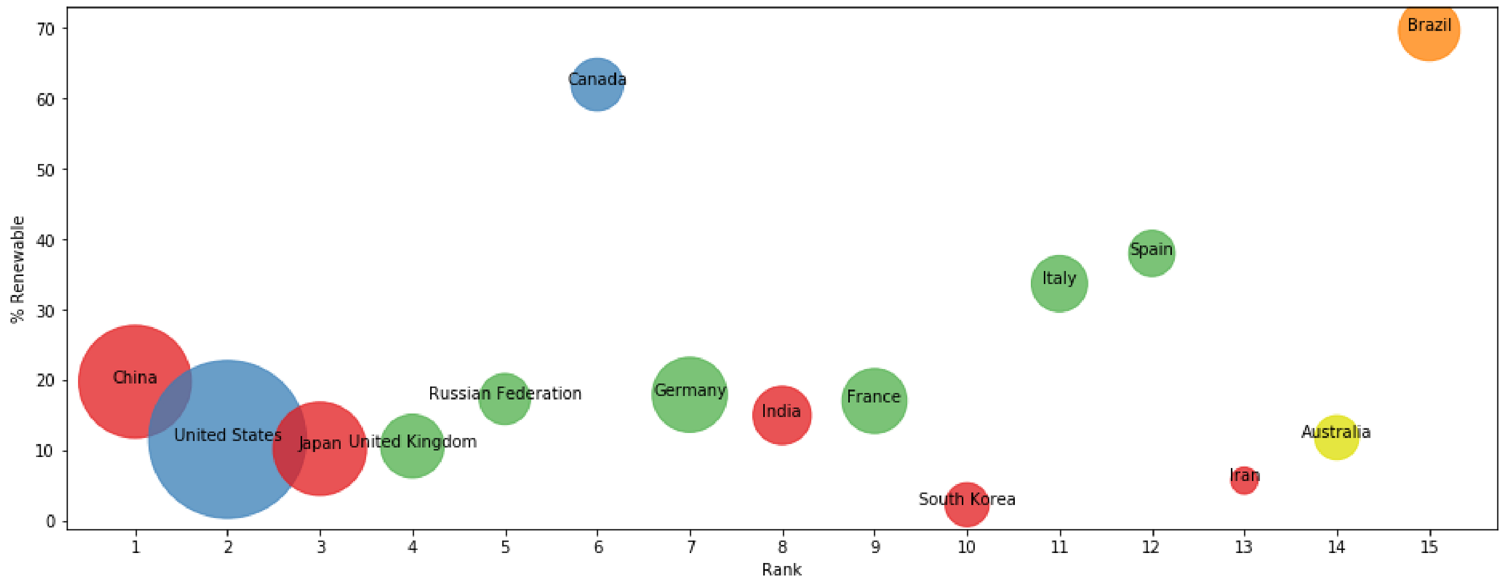
In [163]:

```
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.")
```

In [164]:

```
plot_optional() # Be sure to comment out plot_optional() before submitting the assignment!
```

```
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 count
ries' 2014 GDP, and the color corresponds to the continent.
```



In [ ]:

```
```

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ (https://www.coursera.org/learn/python-data-analysis/resources/0dhYG) course resource.*

In [123]:

```
import pandas as pd
import numpy as np
from scipy.stats import ttest_ind
```

Assignment 4 - Hypothesis Testing

This assignment requires more individual learning than previous assignments - 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. And of course, the discussion forums are open for interaction with your peers and the course staff.

Definitions:

- A *quarter* is a specific three month period, Q1 is January through March, Q2 is April through June, Q3 is July through September, Q4 is October through December.
- A *recession* is defined as starting with two consecutive quarters of GDP decline, and ending with two consecutive quarters of GDP growth.
- A *recession bottom* is the quarter within a recession which had the lowest GDP.
- A *university town* is a city which has a high percentage of university students compared to the total population of the city.

Hypothesis: University towns have their mean housing prices less effected by recessions. Run a t-test to compare the ratio of the mean price of houses in university towns the quarter before the recession starts compared to the recession bottom. (`price_ratio=quarter_before_recession/recession_bottom`)

The following data files are available for this assignment:

- From the Zillow research data site (http://www.zillow.com/research/data/) there is housing data for the United States. In particular the datafile for all homes at a city level (http://files.zillowstatic.com/research/public/City/City_Zhvi_AllHomes.csv), `City_Zhvi_AllHomes.csv` , has median home sale prices at a fine grained level.
- From the Wikipedia page on college towns is a list of university towns in the United States (https://en.wikipedia.org/wiki/List_of_college_towns#College_towns_in_the_United_States) which has been copy and pasted into the file `university_towns.txt` .
- From Bureau of Economic Analysis, US Department of Commerce, the GDP over time (http://www.bea.gov/national/index.htm#gdp) of the United States in current dollars (use the chained value in 2009 dollars), in quarterly intervals, in the file `gdplev.xls` . For this assignment, only look at GDP data from the first quarter of 2000 onward.

Each function in this assignment below is worth 10%, with the exception of `run_ttest()` , which is worth 50%.

In [124]:

```
# Use this dictionary to map state names to two letter acronyms
states = {'OH': 'Ohio', 'KY': 'Kentucky', 'AS': 'American Samoa', 'NV': 'Nevada',
          'WY': 'Wyoming', 'NA': 'National', 'AL': 'Alabama', 'MD': 'Maryland',
          'AK': 'Alaska', 'UT': 'Utah', 'OR': 'Oregon', 'MT': 'Montana',
          'IL': 'Illinois', 'TN': 'Tennessee', 'DC': 'District of Columbia',
          'VT': 'Vermont', 'ID': 'Idaho', 'AR': 'Arkansas', 'ME': 'Maine',
          'WA': 'Washington', 'HI': 'Hawaii', 'WI': 'Wisconsin', 'MI': 'Michigan',
          'IN': 'Indiana', 'NJ': 'New Jersey', 'AZ': 'Arizona', 'GU': 'Guam',
          'MS': 'Mississippi', 'PR': 'Puerto Rico', 'NC': 'North Carolina',
          'TX': 'Texas', 'SD': 'South Dakota', 'MP': 'Northern Mariana Islands',
          'IA': 'Iowa', 'MO': 'Missouri', 'CT': 'Connecticut', 'WV': 'West Virginia',
          'SC': 'South Carolina', 'LA': 'Louisiana', 'KS': 'Kansas', 'NY': 'New York',
          'NE': 'Nebraska', 'OK': 'Oklahoma', 'FL': 'Florida', 'CA': 'California',
          'CO': 'Colorado', 'PA': 'Pennsylvania', 'DE': 'Delaware', 'NM': 'New Mexico',
          'RI': 'Rhode Island', 'MN': 'Minnesota', 'VI': 'Virgin Islands', 'NH': 'New Hampshire',
          'MA': 'Massachusetts', 'GA': 'Georgia', 'ND': 'North Dakota', 'VA': 'Virginia'}
```

In [125]:

```python
import pandas as pd
def get_list_of_university_towns():
    '''Returns a DataFrame of towns and the states they are in from the
    university_towns.txt list. The format of the DataFrame should be:
    DataFrame( [ ["Michigan", "Ann Arbor"], ["Michigan", "Yipsilanti"] ],
    columns=["State", "RegionName"]  )

    The following cleaning needs to be done:

    1. For "State", removing characters from "[" to the end.
    2. For "RegionName", when applicable, removing every character from " (" to the end.
    3. Depending on how you read the data, you may need to remove newline character '\n'. '''

    state_towns = []

    file = open('university_towns.txt')
    data = file.readlines()
    # ['Alabama[edit]\n', 'Auburn (Auburn University)[1]\n'...]

    for item in data:
        # remove spaces
        item = item.strip()
        """
        Alabama[edit]
        Auburn (Auburn University)[1]
        Florence (University of North Alabama)
        ...
        """

        """
        We can observe that every state is followed by [edit]
        """
        if item[-6:] == '[edit]':
            state = item[:-6]
        elif '(' in item:
            pos = item.index('(')
            town = item[:pos-1]
            state_towns.append([state, town])
        else:
            state_towns.append([state, item])

    return pd.DataFrame(state_towns, columns = ['State','RegionName'])

get_list_of_university_towns()
```

Out[125]:

	State	RegionName
0	Alabama	Auburn
1	Alabama	Florence
2	Alabama	Jacksonville
3	Alabama	Livingston
4	Alabama	Montevallo
...	...	...
512	Wisconsin	River Falls
513	Wisconsin	Stevens Point
514	Wisconsin	Waukesha
515	Wisconsin	Whitewater
516	Wyoming	Laramie

517 rows × 2 columns

In [126]:

```
import pandas as pd
def get_recession_start():
    '''Returns the year and quarter of the recession start time as a
    string value in a format such as 2005q3'''

    GDP = pd.read_excel('gdplev.xls', skiprows=7)
    """
            Unnamed: 0  Unnamed: 1  Unnamed: 2  Unnamed: 3 Unnamed: 4  Unnamed: 5  \
    0        1929.0       104.6      1056.6         NaN     1947q1       243.1
    1        1930.0        92.2       966.7         NaN     1947q2       246.3
    2        1931.0        77.4       904.8         NaN     1947q3       250.1
    """

    # We need only quater(Unnamed: 4) and quaterly GDP (Unmamed: 5)
    GDP_Quater = GDP[['Unnamed: 4', 'Unnamed: 5']]
    GDP_Quater.columns = ['Quater', 'GDP']
    # print(GDP_Quater)
    '''
         Quater      GDP
    0    1947q1    243.1
    1    1947q2    246.3
    2    1947q3    250.1
    3    1947q4    260.3
    4    1948q1    266.2
    ..      ...      ...
    273  2015q2  17998.3
    274  2015q3  18141.9
    275  2015q4  18222.8
    276  2016q1  18281.6
    277  2016q2  18450.1
    '''

    # A recession is defined as starting with two consecutive quarters of GDP decline
    # print(GDP_Quater.iloc[0][0])
    # 1947q1
    # print(GDP_Quater.iloc[3][1])
    # 260.3

    end_loop = len(GDP_Quater)-1
    # print(range(end_loop))
    # range(0, 277)

    recession_start = []
    for i in range(end_loop-2):
        if ((GDP_Quater.iloc[i][1] > GDP_Quater.iloc[i+1][1]) &
            (GDP_Quater.iloc[i+1][1] > GDP_Quater.iloc[i+2][1])):
            recession_start.append(GDP_Quater.iloc[i][0])

    # recession_start = ['1948q4', '1953q2', '1953q3', '1957q3', '2008q3', '2008q4']

    # For this assignment, only look at GDP data from the first quarter of 2000 onward.
    return recession_start[4]
```

```
get_recession_start()
```

Out[126]: '2008q3'

In [127]:

```
def get_recession_end():
    '''Returns the year and quarter of the recession end time as a
    string value in a format such as 2005q3'''

    GDP = pd.read_excel('gdplev.xls', skiprows=7)
    GDP_Quater = GDP[['Unnamed: 4', 'Unnamed: 5']]
    GDP_Quater.columns = ['Quater', 'GDP']

    # test = GDP_Quater[GDP_Quater['Quater'] == '2008q3']
    # print(test)
    """
          Quater      GDP
    246  2008q3  14843.0
    """
    GDP_Quater = GDP_Quater[246:]

    # Recession ending with two consecutive quarters of GDP growth
    recession_end = []
    for i in range(len(GDP_Quater) - 3):
        if ((GDP_Quater.iloc[i][1] < GDP_Quater.iloc[i+1][1]) &
            (GDP_Quater.iloc[i+1][1] < GDP_Quater.iloc[i+2][1])):
             # We should return the quater after 2 consecutives quaters of GDP growth
             # which explains i+2
            recession_end.append(GDP_Quater.iloc[i+2][0])

    return recession_end[0]

get_recession_end()
```

Out[127]: '2009q4'

In [128]:

```
def get_recession_bottom():
    '''Returns the year and quarter of the recession bottom time as a
    string value in a format such as 2005q3'''

    GDP = pd.read_excel('gdplev.xls', skiprows=7)
    GDP_Quater = GDP[['Unnamed: 4', 'Unnamed: 5']]
    GDP_Quater.columns = ['Quater', 'GDP']

    # A recession bottom is the quarter within a recession which had the lowest GDP.
    # We know from previous work that the recession starts from 2008q3 and ends in 2009q4
    begin = GDP_Quater[GDP_Quater['Quater'] == '2008q3'].index[0]
    # 246
    end = GDP_Quater[GDP_Quater['Quater'] == '2009q4'].index[0]
    # 251

    recession = GDP_Quater[begin:end+1]
    min_GDP = min(recession['GDP'])
    recession_bottom = recession[recession['GDP'] == min_GDP]
    """
          Quater      GDP
    249  2009q2  14340.4
    """
    return recession_bottom.iloc[0][0]
get_recession_bottom()
```

Out[128]:

```
'2009q2'
```

```
In [129]: def convert_housing_data_to_quarters():
              '''Converts the housing data to quarters and returns it as mean
              values in a dataframe. This dataframe should be a dataframe with
              columns for 2000q1 through 2016q3, and should have a multi-index
              in the shape of ["State","RegionName"].
              '''

              homes = pd.read_csv('City_Zhvi_AllHomes.csv')
              homes['State'] = homes['State'].map(states)
              homes.set_index(['State', 'RegionName'], inplace=True)

              homes = homes.drop(homes.columns[[0] + list(range(0,49))], axis=1)
              # we can also use filter() to select all the data from 2000
              # homes = homes.filter(regex='^20', axis=1)

              # group select columns by quarter, calculates average per quarter
              homes = homes.groupby(pd.PeriodIndex(homes.columns, freq='q'), axis=1).mean()
              # freq = 'Y': return annual mean value

              return homes

          convert_housing_data_to_quarters()
```

Out[129]:

		2000Q1	2000Q2	2000Q3	2000Q4	2001Q1	2001Q2	2001Q3	2001Q4	2002Q1	2002Q2	...	2015Q1	2015Q2	2015Q3
State	**RegionName**														
New York	New York	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	523500.000000	532033.333333	548500.000000
California	Los Angeles	207066.666667	214466.666667	220966.666667	226166.666667	233000.000000	239100.000000	245066.666667	253033.333333	261966.666667	272700.000000	...	526666.666667	535133.333333	545300.000000
Illinois	Chicago	138400.000000	143633.333333	147866.666667	152133.333333	156933.333333	161800.000000	166400.000000	170433.333333	175500.000000	177566.666667	...	194866.666667	198866.666667	201566.666667
Pennsylvania	Philadelphia	53000.000000	53633.333333	54133.333333	54700.000000	55333.333333	55533.333333	56266.666667	57533.333333	59133.333333	60733.333333	...	116700.000000	117900.000000	120633.333333
Arizona	Phoenix	111833.333333	114366.666667	116000.000000	117400.000000	119600.000000	121566.666667	122700.000000	124300.000000	126533.333333	128366.666667	...	173266.666667	176500.000000	180566.666667
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Wisconsin	Town of Wrightstown	101766.666667	105400.000000	111366.666667	114866.666667	125966.666667	129900.000000	129900.000000	129433.333333	131900.000000	134200.000000	...	148866.666667	150866.666667	152500.000000
New York	Urbana	79200.000000	81666.666667	91700.000000	98366.666667	94866.666667	98533.333333	102966.666667	98033.333333	93966.666667	94600.000000	...	131166.666667	132233.333333	131066.666667
Wisconsin	New Denmark	114566.666667	119266.666667	126066.666667	131966.666667	143800.000000	146966.666667	148366.666667	149166.666667	153133.333333	156733.333333	...	182733.333333	185166.666667	184433.333333
California	Angels	151000.000000	155900.000000	158100.000000	167466.666667	176833.333333	183766.666667	190233.333333	184566.666667	184033.333333	186133.333333	...	230233.333333	228733.333333	240166.666667
New Jersey	Lebanon Borough	165800.000000	169833.333333	173266.666667	177233.333333	180333.333333	183800.000000	188266.666667	191866.666667	193366.666667	200800.000000	...	232100.000000	231533.333333	232000.000000

10830 rows × 70 columns

In [130]:

```
def run_ttest():
    '''First creates new data showing the decline or growth of housing prices
    between the recession start and the recession bottom. Then runs a ttest
    comparing the university town values to the non-university towns values,
    return whether the alternative hypothesis (that the two groups are the same)
    is true or not as well as the p-value of the confidence.

    Return the tuple (different, p, better) where different=True if the t-test is
    True at a p<0.01 (we reject the null hypothesis), or different=False if
    otherwise (we cannot reject the null hypothesis). The variable p should
    be equal to the exact p value returned from scipy.stats.ttest_ind(). The
    value for better should be either "university town" or "non-university town"
    depending on which has a lower mean price ratio (which is equivilent to a
    reduced market loss).'''

    # Run a t-test to compare the ratio of the mean price of houses in university towns
    # the quarter before the recession starts compared to the recession bottom.

    homes = convert_housing_data_to_quarters()
    recession_start = get_recession_start().upper()
    recession_bottom = get_recession_bottom().upper()
    university_towns = get_list_of_university_towns()

    recession_before_index = homes.columns.get_loc(recession_start) - 1
    # print(recession_before_index)
    # 33
    recession_before = homes.columns[recession_before_index]
    # Period('2008Q2', 'Q-DEC')

    ratio = pd.DataFrame({'ratio': homes[recession_before].div(homes[recession_bottom])})
    # homes['ratio'] = homes[recession_before].div(homes[recession_bottom])

    # homes.columns = homes.columns.to_series().astype(str)
    """
    Index(['2000Q1', '2000Q2', '2000Q3', '2000Q4', '2001Q1', '2001Q2', '2001Q3',
       '2001Q4', '2002Q1', '2002Q2', '2002Q3', '2002Q4', '2003Q1', '2003Q2',
       '2003Q3', '2003Q4', '2004Q1', '2004Q2', '2004Q3', '2004Q4', '2005Q1',
       '2005Q2', '2005Q3', '2005Q4', '2006Q1', '2006Q2', '2006Q3', '2006Q4',
       '2007Q1', '2007Q2', '2007Q3', '2007Q4', '2008Q1', '2008Q2', '2008Q3',
       '2008Q4', '2009Q1', '2009Q2', '2009Q3', '2009Q4', '2010Q1', '2010Q2',
       '2010Q3', '2010Q4', '2011Q1', '2011Q2', '2011Q3', '2011Q4', '2012Q1',
       '2012Q2', '2012Q3', '2012Q4', '2013Q1', '2013Q2', '2013Q3', '2013Q4',
       '2014Q1', '2014Q2', '2014Q3', '2014Q4', '2015Q1', '2015Q2', '2015Q3',
       '2015Q4', '2016Q1', '2016Q2', '2016Q3', '2016Q4', '2017Q1', '2017Q2'],
      dtype='object')

    """
    # homes = pd.concat([homes, ratio], axis=1)

    is_univ_town = pd.merge(university_towns, ratio, how='inner', on=['State', 'RegionName'])
    is_univ_town['IsUnivTown'] = True
    is_univ_town = pd.merge(is_univ_town, ratio, how='outer', on=['State', 'RegionName', 'ratio'])
    is_univ_town['IsUnivTown'] = is_univ_town['IsUnivTown'].fillna(False)
    is_univ_town.set_index(['State', 'RegionName'], inplace=True)
    is_univ_town = is_univ_town.sort_index(level=(0, 1))

    univ_town = is_univ_town[is_univ_town['IsUnivTown'] == True]
    non_univ_town = is_univ_town[is_univ_town['IsUnivTown'] == False]

    tstat, p = (ttest_ind(univ_town['ratio'].dropna(), non_univ_town['ratio'].dropna()))

    # different=True if the t-test is True at a p<0.01
    different = True if p<0.01 else False
    better = ('University Town' if univ_town['ratio'].dropna().mean() < non_univ_town['ratio'].dropna().mean()
                                else 'Non-University Town')

    return (different, p, better)

run_ttest()
```

```
Out[130]: (True, 0.009884030627156846, 'University Town')
```

In [131]:

```
def convert_housing_data_to_quarters_alter():
    '''Converts the housing data to quarters and returns it as mean
    values in a dataframe. This dataframe should be a dataframe with
    columns for 2000q1 through 2016q3, and should have a multi-index
    in the shape of ["State","RegionName"].

    Note: Quarters are defined in the assignment description, they are
    not arbitrary three month periods.

    The resulting dataframe should have 67 columns, and 10,730 rows.
    '''
#     a = list(range(3,51))
    df = pd.read_csv('City_Zhvi_AllHomes.csv')
    df = df.drop(df.columns[[0] + list(range(3,51))], axis=1)
#     df2 = df.set_index(['State', 'RegionName'])
    df2 = pd.DataFrame(df[['State', 'RegionName']])
#     print(df2)
    for year in range(2000, 2016):
        df2[str(year) + 'q1'] = df[[str(year) + '-01', str(year) + '-02', str(year) + '-03']].mean(axis = 1)
        df2[str(year) + 'q2'] = df[[str(year) + '-04', str(year) + '-05', str(year) + '-06']].mean(axis = 1)
        df2[str(year) + 'q3'] = df[[str(year) + '-07', str(year) + '-08', str(year) + '-09']].mean(axis = 1)
        df2[str(year) + 'q4'] = df[[str(year) + '-10', str(year) + '-11', str(year) + '-12']].mean(axis = 1)
    year = 2016
    df2[str(year) + 'q1'] = df[[str(year) + '-01', str(year) + '-02', str(year) + '-03']].mean(axis = 1)
    df2[str(year) + 'q2'] = df[[str(year) + '-04', str(year) + '-05', str(year) + '-06']].mean(axis = 1)
    df2[str(year) + 'q3'] = df[[str(year) + '-07', str(year) + '-08']].mean(axis = 1)
#     df2 = df2.set_index(['State', 'RegionName'])
    df2['State'] = [states[state] for state in df2['State']]
    df2 = df2.set_index(['State', 'RegionName'])
    ans = pd.DataFrame(df2)

#     print(ans)
    return ans
convert_housing_data_to_quarters_alter()
# convert_housing_data_to_quarters().loc["Texas"].loc["Austin"].loc["2010q3"]
```

Out[131]:

		2000q1	2000q2	2000q3	2000q4	2001q1	2001q2	2001q3	2001q4	2002q1	2002q2	...	2014q2	2014q3	2014q4
State	RegionName														
New York	New York	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	515333.333333	519100.000000	522166.666667
California	Los Angeles	207066.666667	214466.666667	220966.666667	226166.666667	233000.000000	239100.000000	245066.666667	253033.333333	261966.666667	272700.000000	...	498400.000000	509133.333333	517866.666667
Illinois	Chicago	138400.000000	143633.333333	147866.666667	152133.333333	156933.333333	161800.000000	166400.000000	170433.333333	175500.000000	177566.666667	...	188133.333333	190266.666667	193733.333333
Pennsylvania	Philadelphia	53000.000000	53633.333333	54133.333333	54700.000000	55333.333333	55533.333333	56266.666667	57533.333333	59133.333333	60733.333333	...	114633.333333	115866.666667	116600.000000
Arizona	Phoenix	111833.333333	114366.666667	116000.000000	117400.000000	119600.000000	121566.666667	122700.000000	124300.000000	126533.333333	128366.666667	...	166333.333333	167533.333333	170466.666667
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Wisconsin	Town of Wrightstown	101766.666667	105400.000000	111366.666667	114866.666667	125966.666667	129900.000000	129900.000000	129433.333333	131900.000000	134200.000000	...	147966.666667	148300.000000	147466.666667
New York	Urbana	79200.000000	81666.666667	91700.000000	98366.666667	94866.666667	98533.333333	102966.666667	98033.333333	93966.666667	94600.000000	...	126466.666667	125633.333333	128666.666667
Wisconsin	New Denmark	114566.666667	119266.666667	126066.666667	131966.666667	143800.000000	146966.666667	148366.666667	149166.666667	153133.333333	156733.333333	...	168400.000000	174800.000000	179500.000000
California	Angels	151000.000000	155900.000000	158100.000000	167466.666667	176833.333333	183766.666667	190233.333333	184566.666667	184033.333333	186133.333333	...	214600.000000	217966.666667	222833.333333
New Jersey	Lebanon Borough	165800.000000	169833.333333	173266.666667	177233.333333	180333.333333	183800.000000	188266.666667	191866.666667	193366.666667	200800.000000	...	233100.000000	232000.000000	229300.000000

10830 rows × 67 columns

In [ ]: