Macroeconomic data visualization with Python

Manual

09/2016

Chia-Chien, Hung

Xiao-Ya, Lai

1. Introduction

This is the manual for our project: **Macroeconomic data visualization with Python**. In this manual we will first guide you how to build the environment before starting to compile the python file. And then we will show you the scripts and the results after compiling the python file. Ready to start.

2. Building the environment

(1) Install Python

The programming language we use is **Python**. You need to download at least version **2.7.11** (See more on: https://www.python.org/downloads/). And also make sure that the version you download is compatible with your environment (Windows, Linux, Mac IOS).

And also for text editor, you can download **Pycharm** (See more on: https://www.jetbrains.com/pycharm/download/).

(2) Install the packages

Python have many useful packages and you can download the packages on **PyPI** (See more on: https://pypi.python.org/pypi).

The packages we use in this project are as following:

csv	Import and export format for spreadsheets and databases
scipy	Scientific computing with a collection of packages
pandas	Provide rich data structures for the efficiency of data working environment
BeautifulSoup	Pull data out of HTML and XML files and do navigating, searching, and modifying on
PIL	Provide powerful image processing and graphics capabilities
svgwrite	Create SVG drawings
svgutils	An utility package that helps to edit and concatenate SVG files
string	Contain a number of useful constants and classes on string
matplotlib	Produce plots and other 2D data visualizations
numpy	Scientific computing with a collection of packages
PySide	Python bindings for the Qt cross-platform application and UI framework
os	Provide a portable way of using operating system dependent functionality
shutil	Offer a number of high-level operations on files and collections of files
Images2gif	Read and write animated gifs

There might be many versions of a single package. You need to make sure

that the latest version really fit the computer environment (Windows, Linux, Mac IOS) and could be compatible with the version of the python you installed. Otherwise, you should choose the former version of the package instead of the latest version.

(a) Case1

If you download the package in **.whl** file, you need to first open a console and cd to the folder where you save.

>>cd C:\some-packagepath

And use

>>pip install some-package.whl

If pip is not found in path, use

>>python -m pip install some-package.whl

(b) Case2

If you download the package in **.tar.gz** or **.zip** file, you need to unpack the tar.gz file into a folder.

And then open a console and cd to the folder.

>>cd C:\some-packagepath

And use the following code to successfully install

>>python setup.py install

(c) Case3

If you download the package in **.exe** file, you just need to run the **.exe** file and choose the compatible environment.

(3) Build the dataset

(a) worlddata.csv

We save our database in .csv format. The open sources we use are the publication from IMF (International Monetary Fund) **The World Economic Outlook (WEO) April 2016** and **Wikipedia**. We selected the yearly dataset from 2006 to 2015 within 190 countries. We did the Data Cleaning and rearranged our data into the csv file in order for Python to read the file. The data are separating into two parts: countries' basic information and macroeconomic yearly data.

For countries' basic information, the titles include **Country** (the name of the countries), **Country Code** (Alpha-2 code), **Alpha-3 code**, **Capital**, **Continent** (categorized into 6 continents: Africa, Asia, Europe, North America, South America and Oceania), **Area** (size of geographical area) and **Telephone Code** (Country calling code). And for macroeconomic yearly data (2006~2015), the titles include **Year** (the specific year in which the macroeconomic data are collected), **GDP percapita** (U.S. dollars),

Unemployment Rate (%), **Population** (Millions), **Government Net Debt** (per GDP).

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Country by	Country	Country Co	ISO	Capital	Continent	Telephone	Area	Year	GDP perca	Unemployr	Population	Government Net
2	Afghanista	Afghanista	AF	AFG	Kabul	Asia	93	647500	2015	653.6	n/a	32.01	6.824
3	Albania	Albania	AL	ALB	Tirana	Europe	355	28748	2015	4594.91	17.1	2.89	71.886
4	Algeria	Algeria	DZ	DZA	Algiers	Africa	213	2381740	2015	5458.87	11.27	39.9	-3.226
5	Angola	Angola	AO	AGO	Luanda	Africa	244	1246700	2015	5199.26	n/a	25.12	n/a
6	Antigua an	Antigua an	AG	ATG	Saint John	North Ame	1 268	443	2015	14126.16	n/a	0.09	n/a
7	Argentina	Argentina	AR	ARG	Buenos Air	South Ame	54	2766890	2015	12774.42	6.47	43.1	n/a
8	Armenia	Armenia	AM	ARM	Yerevan	Asia	374	29800	2015	3900.7	17.7	2.99	n/a
9	Australia	Australia	AU	AUS	Canberra	Oceania	61	7686850	2015	61062.61	6.08	24.02	17.902
10	Austria	Austria	AT	AUT	Vienna	Europe	43	83858	2015	51433	5.73	8.56	47.829
11	Azerbaijan	Azerbaijan	AZ	AZE	Baku	Asia	994	86600	2015	8055.21	6.05	9.42	n/a
12	Bahrain	Bahrain	BH	BHR	Manama	Asia	973	665	2015	26686.27	n/a	1.29	63.306
13	Bangladesh	Bangladesh	BD	BGD	Dhaka	Asia	880	144000	2015	1161.85	n/a	159.86	n/a
14	Barbados	Barbados	BB	BRB	Bridgetowr	North Ame	1	431	2015	15596.53	12.16	0.28	75.415
15	Belarus	Belarus	BY	BLR	Minsk	Europe	375	207600	2015	8041.75	1	9.5	n/a
16	Belgium	Belgium	BE	BEL	Brussels	Europe	32	30510	2015	47517.97	8.33	11.34	63.819
17	Belize	Belize	BZ	BLZ	Belmopan	North Ame	501	22966	2015	4760.28	11.96	0.36	76.258
18	Benin	Benin	BJ	BEN	Porto-Novo	Africa	229	112620	2015	905.58	n/a	10.86	n/a
19	Bhutan	Bhutan	BT	BTN	Thimphu	Asia	975	47000	2015	2590.6	3.2	0.78	n/a
20	Bolivia	Bolivia (Pl	BO	BOL	La Paz	South Ame	591	1098580	2015	2943.46	4	11.51	25.68
21	Bosnia and	Bosnia and	BA	BIH	Sarajevo	Europe	387	51129	2015	4784.82	27.7	3.86	39.084
22	Botswana	Botswana	BW	BWA	Gaborone	Africa	267	600370	2015	7548.19	n/a	2.13	n/a

(b) GDP.csv

Same as the source of “worlddata.csv”, we get the data of world GDP from IMF (International Monetary Fund) **The World Economic Outlook (WEO) April 2016** and **Wikipedia**. The original world data file contains a country’s basic information and also yearly data from 1980 to 2015, and some has predictions for 2016 and later. For deep searching, we extract the macroeconomic data separately from the original table, and form a new .csv file. In the file, we delete the unnecessary information and only reserve the needed ones, including merely year and countries’ data from 1985 to 2015. From this kind of data file, we can get a time series figure showing the change s and trends of GDP in these countries.

Gross domestic product (GDP) is a monetary measure of the market value of all final goods and services produced in a period, and the data we get here take one year as a period. Nominal GDP estimates we use here are the kind of data that is commonly used to determine the economic performance of a whole country or region, and to make international comparisons. This GDP.csv file contains all the countries included in the original IMF data file, from Afghanistan to Zimbabwe. And all the data ordered by year. What must be mentioned is that, there might be some data lost or not collected and presented as “n/a” in the data, but this kind of “default” data may obstruct our visualization, so we substitute them with zero.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	DATE	Afghanist	Albania	Algeria	Angola	Antigua a	Argentina	Armenia	Australia	Austria	Azerbaija	The Bahar	Bahrain
2	1980	0	2008.51	4620.99	1782.18	4044.06	6282.1	0	10437.28	11201.82	0	9334.41	17726.54
3	1981	0	2275.89	5047.2	1813.83	4605.31	6360.37	0	11695.23	12210.99	0	9724.77	19155.22
4	1982	0	2435.6	5525.98	1876.72	4900.21	6433.77	0	12225.65	13199.46	0	10728.16	20888.88
5	1983	0	2508.45	5861.89	1980.87	5490.66	6840.47	0	12489.1	14146.94	0	11644.12	22466.12
6	1984	0	2595.83	6210.35	2118.24	6378.87	7103.86	0	13583.01	14696.02	0	12077.16	23447.22
7	1985	0	2584.55	6455.51	2007.84	7197.64	6706.95	0	14574.94	15496.07	0	12806.87	23174.19
8	1986	0	2730.04	6399.53	2049.17	8087.05	7238.08	0	15007.96	16161.6	0	13177.56	22946.23
9	1987	0	2723.23	6349.82	2132.77	9042.87	7524.83	0	15892.14	16830.16	0	13780.28	22443.55
10	1988	0	2724.2	6259.97	2280.94	10015.83	7543.65	0	16847.49	17564.35	0	14286.67	23757.39
11	1989	0	3048.13	6650	2307.34	11141.08	7198.66	0	18037.71	18911.72	0	14927.69	24063.96
12	1990	0	2846.94	6861.68	2250.07	11884.21	7213.37	0	18730.13	20328.26	0	15383.42	25605.09
13	1991	0	2155.83	6835.35	2279.62	12503.69	8126.03	0	18911.02	21541.54	0	14930.25	25979.8
14	1992	0	2066.21	6933.24	2131.52	12708.39	9043.85	1416.27	19653.59	22240.08	4558.27	14435.31	28000.67
15	1993	0	2329.82	6787.96	1610.54	13419.72	9693.49	1275.76	20726.58	22646.35	3575.25	14570.72	30045.6
16	1994	0	2656.17	6719.63	1618.3	14207.04	10344.72	1406.67	21982.67	23545.92	2890.55	15086.97	30848.12
17	1995	0	2949.17	6980.53	1844.52	13408.39	10134.31	1585.08	22818.64	24632.3	2523.51	15750.07	31263.42
18	1996	0	3248.64	7247.34	2070.46	14094.78	10760.72	1724.22	23951.32	25652.4	2622.11	16432.09	32014.28
19	1997	0	2964.64	7329.61	2193.3	14786.35	11696.85	1830.47	25163.65	26627.1	2876.15	17248.13	32460.01
20	1998	0	3287.41	7664.91	2253.5	15300.4	12142.25	1985.33	26340.43	27852.9	3052.8	17959.15	33508.8
21	1999	0	3812.42	7908.49	2269.81	15826.27	11782.05	2093.08	27580.17	29252.55	3408.37	19222.76	30627.84
22	2000	0	4184.05	8246.9	2322.68	16393.02	11829.84	2273.35	28768.7	30849.74	3647.12	20152.29	37435.34
23	2001	0	4662.76	8562.15	2403.43	15759.52	11450.11	2561.94	29774.59	31904.4	3926.48	20840.11	37833.86
24	2002	845.3	4949.49	9046.01	2693.03	16232.33	10260.04	2996.23	31100.27	32756.21	4262.12	21419.33	36950.02

(c) CPI.csv

Same as “GDP.csv”, we get these CPI data of world GDP from IMF (International Monetary Fund) **The World Economic Outlook (WEO) April 2016** and **Wikipedia**. The original world data file contains a country’s basic information and also yearly data from 1980 to 2015, and some has predictions for 2016 and later. As CPI can serve as a better predictor for future trend, we did not delete the predictions after 2016, so in this file, there are data from 1980 to 2021. And these data are also included in our data visualization. From this kind of data file, we can get a time series figure showing the change s and trends of CPI in these countries.

CPI, **consumer price index (CPI)** measures changes in the price level of a market basket of consumer goods and services purchased by households. This CPI.csv file also contains all the countries included in the original IMF data file, from Afghanistan to Zimbabwe ordered by time. What must be mentioned is that, there might be some data lost or not collected and presented as “n/a” in the data, but this kind of “default” data may obstruct our visualization, so we still substitute them with zero.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	DATE	Afghanist	Afghanist	Albania	Albania	Algeria	Algeria	Angola	Angola	Antigua a	Antigua a	Argentina	Argentir
2	1980	0	0	0	0	8.975	9.668	0	46.708	48.819	18.999	0	0
3	1981	0	0	0	0	10.286	14.61	0	1.391	54.416	11.465	0	0
4	1982	0	0	0	0	10.964	6.593	0	1.833	56.685	4.17	0	0
5	1983	0	0	0	0	11.823	7.835	0	1.833	58.015	2.346	0	0
6	1984	0	0	0	0	12.569	6.31	0	1.833	60.245	3.844	0	0
7	1985	0	0	0	0	13.88	10.432	0	1.833	60.845	0.996	0	0
8	1986	0	0	0	0	15.824	14.007	0	1.833	61.149	0.5	0	0
9	1987	0	0	0	0	16.751	5.857	0	1.833	63.352	3.602	0	0
10	1988	0	0	0	0	17.746	5.938	0	1.833	67.66	6.8	0	0
11	1989	0	0	4.043	0	19.373	9.172	0	1.833	70.611	4.362	0	0
12	1990	0	0	4.036	-0.18	21.17	9.272	0	1.833	75.265	6.591	0	0
13	1991	0	0	5.478	35.717	26.652	25.9	0	85.265	78.666	4.518	0	0
14	1992	0	0	17.857	225.996	35.101	31.7	0	299.097	81.026	3	0	0
15	1993	0	0	33.035	85.005	42.297	20.5	0	1379.48	83.538	3.1	0	0
16	1994	0	0	40.49	22.565	54.563	29	0	949.771	88.955	6.485	0	0
17	1995	0	0	43.645	7.793	70.823	29.8	0.001	2672.23	91.39	2.737	0	0
18	1996	0	0	49.203	12.734	84.067	18.7	0.027	4146.01	94.1	2.965	0	0
19	1997	0	0	65.522	33.166	88.859	5.7	0.088	221.492	94.441	0.362	26.468	0
20	1998	0	0	79.05	20.646	93.258	4.95	0.183	107.429	97.521	3.262	26.713	0.925
21	1999	0	0	79.358	0.39	95.682	2.6	0.637	248.248	98.628	1.135	26.401	-1.167
22	2000	0	0	79.389	0.039	95.969	0.3	2.707	325.029	98.469	-0.162	26.153	-0.939
23	2001	0	0	81.867	3.121	100	4.2	6.838	152.586	100.374	1.935	25.875	-1.065
24	2002	37.847	0	86.139	5.218	101.43	1.43	14.284	108.893	102.791	2.408	32.568	25.869
25	2003	51.345	35.663	88.158	2.344	105.75	4.259	28.313	98.219	104.84	1.994	36.946	13.443

(d) Population.csv

As the two files before, this Population.csv file is also extracted from the original file, and population refers to the total amount of people in one country.

	A	B	C	D	E	F	G	H	I	J	K	L
1	DATE	Afghanist	Albania	Algeria	Angola	Antigua a	Argentina	Armenia	Australia	Austria	Azerbaija	The Baha
2	1980	0	2.76	18.67	8.39	0.07	27.95	0	14.8	7.54	0	0.21
3	1981	0	2.82	19.25	8.62	0.07	28.45	0	15.04	7.56	0	0.22
4	1982	0	2.88	19.86	8.84	0.07	28.93	0	15.29	7.57	0	0.22
5	1983	0	2.94	20.52	9.07	0.07	29.34	0	15.48	7.54	0	0.23
6	1984	0	3	21.18	9.31	0.07	29.84	0	15.68	7.54	0	0.23
7	1985	0	3.06	22.2	10.5	0.06	30.35	0	15.9	7.55	0	0.23
8	1986	0	3.12	22.8	10.8	0.06	30.74	0	16.14	7.56	0	0.24
9	1987	0	3.18	23.4	11.07	0.06	31.09	0	16.4	7.57	0	0.24
10	1988	0	3.25	24.1	11.37	0.06	31.47	0	16.69	7.58	0	0.25
11	1989	0	3.31	24.7	11.68	0.06	31.86	0	16.94	7.59	0	0.25
12	1990	0	3.31	25.02	12	0.06	32.53	0	17.17	7.65	0	0.25
13	1991	0	3.25	25.64	12.36	0.06	32.97	0	17.38	7.71	0	0.26
14	1992	0	3.22	26.27	12.73	0.06	33.42	3.45	17.56	7.8	7.46	0.26
15	1993	0	3.2	26.89	13.11	0.06	33.92	3.37	17.72	7.88	7.49	0.27
16	1994	0	3.14	27.5	13.5	0.07	34.35	3.29	17.89	7.93	7.6	0.27
17	1995	0	3.14	28.06	13.91	0.07	34.78	3.22	18.12	7.94	7.73	0.28
18	1996	0	3.17	28.57	14.32	0.07	35.2	3.17	18.33	7.95	7.76	0.28
19	1997	0	3.15	29.05	14.75	0.07	35.6	3.14	18.51	7.97	7.84	0.29
20	1998	0	3.13	29.51	15.2	0.07	36.01	3.11	18.71	7.97	7.91	0.29
21	1999	0	3.11	29.97	15.65	0.07	36.4	3.09	18.92	7.98	8.02	0.3
22	2000	0	3.09	30.51	16.12	0.08	36.78	3.08	19.14	8	8.14	0.3
23	2001	0	3.06	30.95	16.6	0.08	37.16	3.06	19.39	8.02	8.23	0.31
24	2002	22.2	3.05	31.41	17.1	0.08	37.52	3.05	19.61	8.06	8.33	0.31
25	2003	23.12	3.04	31.89	17.62	0.08	37.87	3.04	19.83	8.1	8.42	0.32

(e) Net_debt.csv

Gross net debt here demonstrates a country's fiscal situation, with net debt being high referring to a country's terrible fiscal condition. However, as merely an estimator, we cannot make any affirmative conclusion from the trend or number of it, but we are still able to predict or tell something from these data.

There are also "n/a" in the file, we also replace them with 0 to get a figure successfully.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	DATE	Afghanist	Afghanist	Albania	Albania	Algeria	Algeria	Angola	Angola	Antigua a	Antigua a	Argentina	Argentina
2	1980	0	0	0	0	0	0	0	0	0	0	0	0
3	1981	0	0	0	0	0	0	0	0	0	0	0	0
4	1982	0	0	0	0	0	0	0	0	0	0	0	0
5	1983	0	0	0	0	0	0	0	0	0	0	0	0
6	1984	0	0	0	0	0	0	0	0	0	0	0	0
7	1985	0	0	0	0	0	0	0	0	0	0	0	0
8	1986	0	0	0	0	0	0	0	0	0	0	0	0
9	1987	0	0	0	0	0	0	0	0	0	0	0	0
10	1988	0	0	0	0	0	0	0	0	0	0	0	0
11	1989	0	0	0	0	0	0	0	0	0	0	0	0
12	1990	0	0	0	0	0	0	0	0	1.22	98.07	0	0
13	1991	0	0	0	0	671.03	77.83	0	0	1.29	99.08	0	0
14	1992	0	0	0	0	676.26	62.93	0	0	1.29	95.48	63.25	23.31
15	1993	0	0	0	0	880.73	74.03	0	0	1.3	90.15	71.11	25.11
16	1994	0	0	0	0	1463.25	98.38	0	0	1.43	89.66	81.82	26.54
17	1995	0	0	0	0	2329.7	116.2	0	0	1.53	98.32	88.71	28.68
18	1996	0	0	0	0	2522.4	98.15	0	0	1.57	91.67	99.05	30.39
19	1997	0	0	242.3	69.99	1942.11	69.86	0	0	1.6	87.02	103.72	29.57
20	1998	0	0	277.5	67.81	2062.39	72.86	0	0	2.02	102.61	114.13	31.87
21	1999	0	0	306.38	64.97	2664.14	82.02	0	0	2.16	104.6	123.37	36.3
22	2000	0	0	323.75	61.9	2590.11	62.81	95.83	104.54	2.35	110.84	129.75	38.08
23	2001	0	0	341.52	58.54	2314.17	54.75	201.16	102.05	2.55	122.03	144.22	44.76
24	2002	502.27	280.98	391.31	62.84	2326.1	51.43	385.97	70.95	2.78	128.17	513.99	137.72
25	2003	601.67	273.47	408.3	58.83	2224.65	42.36	727.17	68.69	2.89	127.52	523.79	116.39
26	2004	599.37	243.44	423.96	56.45	2166.59	35.23	852.47	51.95	2.98	122.99	568.15	106.03

(f) Employment.csv

Employment rate always appears with unemployment rate, both exhibiting a country's economic situation by employment condition. However, there might exist some differences among the calculating methods used by each country, so we get these two types of data at the same time and hope to get some comparison results.

There are also default values in the file, we also replace them with 0. What must to be mentioned here is that there are several countries not having the employment rate data, just like China, and when this happens, we will get a blank figure containing nothing.

	A	B	C	D	E	F	G	H	I	J
1	DATE	Afghanist	Albania	Algeria	Angola	Antigua a	Argentina	Armenia	Australia	Austria
2	1980								6.287	2.789
3	1981								6.416	2.799
4	1982								6.418	2.766
5	1983								6.301	2.735
6	1984								6.494	2.745
7	1985								6.701	2.76
8	1986								6.974	2.78
9	1987								7.128	2.785
10	1988								7.39	2.81
11	1989								7.718	2.862
12	1990								7.856	2.929
13	1991								7.667	2.997
14	1992								7.63	3.056
15	1993								7.655	3.055
16	1994								7.897	3.071
17	1995								8.186	3.068
18	1996								8.297	3.047
19	1997								8.363	3.056
20	1998								8.515	3.077
21	1999								8.67	3.108
22	2000								8.899	3.134

(g) Unemployment_rate.csv

Unemployment rate is calculated to exhibit a country's economic situation. When financial crisis comes, a country's unemployment rate will drop significantly and rapidly. So from the figure drawn from this file of data, we will vividly see a country's change in economic situation, and be aware of

the big impact financial crisis has on each country.

There are also default values in the file as other five, and we also replace them by zero.

	A	B	C	D	E	F	G	H	I	J	K	L
1	DATE	Afghanist	Albania	Algeria	Angola	Antigua a	Argentina	Armenia	Australia	Austria	Azerbaija	The Baham
2	1980		5.028	15.789			3	0	6.133	1.6	0	0
3	1981		4.224	15.385			5	0	5.783	2.2	0	0
4	1982		2.813	15			4.5	0	7.183	3.1	0	0
5	1983		3.335	14.286			5	0	9.967	3.7	0	0
6	1984		4.41	16.536			5	0	8.967	3.8	0	0
7	1985		5.853	16.901			6.25	0	8.258	3.6	0	0
8	1986		5.43	18.356			6.3	0	8.117	3.1	0	12.2
9	1987		5.164	20.056			6	0	8.108	3.8	0	11.5
10	1988		6.033	21.801			6.5	0	7.208	2.676	0	11
11	1989		6.721	18.1			8	0	6.175	2.348	0	11.7
12	1990		8.457	19.757			7.6	0	6.95	2.723	0	12
13	1991		8.9	20.263			6.48	0	9.608	3.151	0	12.3
14	1992		26.5	21.368			7.112	0	10.75	3.29	0	14.8
15	1993		22.3	23.152			11.606	0	10.875	3.958	0	13.1
16	1994		18.4	24.362			13.348	0	9.717	3.85	0	13.292
17	1995		12.9	28.105			18.904	0	8.467	3.917	0	10.9
18	1996		12.3	27.986			18.76	0	8.5	4.333	0	11.505
19	1997		14.9	27.961			16.808	0	8.358	4.367	0	9.779
20	1998		17.7	28.021			14.789	0	7.683	4.483	0	7.743
21	1999		18.4	29.293			16.061	0	6.858	3.933	0	7.796
22	2000		16.8	29.496			17.134	0	6.275	3.883	0	7
23	2001		16.44	27.306			19.209	38.4	6.767	4.008	0	6.9
24	2002		15.751	25.664			22.45	35.3	6.367	4.392	0	9.102
25	2003		15	23.716			17.25	31.2	5.917	4.792	9.665	10.835

(h) Worldmap.svg

This is a blank world map in SVG format which can be freely downloaded from Amcharts (https://www.amcharts.com/svg-maps/).

3. Scripts and Results

(1) Country Profile and the Ranking.py

Input and display our data file

Script

```
import csv
from scipy.stats import rankdata
import pandas as pd
#Set the format for displaying the table of our data file
pd.set_option('display.height', 2000)
pd.set_option('display.max_rows', 2000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 200)

f = pd.read_csv('worlddata.csv', dtype=str)
```

Result

```
>>>print f
```

	Country	Country Code	ISO	Capital	Continent	Telephone Code	Area	Year	GDP percapita	Unemployment Rate	Population	Government Net Debt
0	Afghanistan	AF	AFG	Kabul	Asia	93	647500	2015	653.6	n/a	32.01	6.824
1	Albania	AL	ALB	Tirana	Europe	355	28748	2015	4594.91	17.1	2.89	71.886
2	Algeria	DZ	DZA	Algiers	Africa	213	2381740	2015	5458.87	11.27	39.9	-3.226
3	Angola	AO	AGO	Luanda	Africa	244	1246700	2015	5199.26	n/a	25.12	n/a
4	Antigua and Barbuda	AG	ATG	Saint John	North America	1 268	443	2015	14126.16	n/a	0.09	n/a
5	Argentina	AR	ARG	Buenos Aires	South America	54	2766890	2015	12774.42	6.47	43.1	n/a
6	Armenia	AM	ARM	Yerevan	Asia	374	29800	2015	3900.7	17.7	2.99	n/a
7	Australia	AU	AUS	Canberra	Oceania	61	7686850	2015	61062.61	6.08	24.02	17.902
8	Austria	AT	AUT	Vienna	Europe	43	83858	2015	51433	5.73	8.56	47.829
9	Azerbaijan	AZ	AZE	Baku	Asia	994	86600	2015	8055.21	6.05	9.42	n/a
10	Bahrain	BH	BHR	Manama	Asia	973	665	2015	26686.27	n/a	1.29	63.306

(skip)

Build our dataset dictionary

Script

```
def data_dictionary():
    reader = csv.reader(open('worlddata.csv'), delimiter=",")
    #skip header and convert data to map
    headers = reader.next()
    column = {}
    for h in headers:
        column[h] = []
    for row in reader:
        for h, v in zip(headers, row):
            column[h].append(v)
    return column
```

Result

```
>>> print sorted(data_dictionary().keys())
['Alpha-3 code', 'Area', 'Capital', 'Continent', 'Country', 'Country Code', 'Country by IMC', 'GDP
percapita', 'Government Net Debt', 'Population', 'Telephone Code', 'Unemployment Rate', 'Year']
```

Build a ranking method

Script

```
def ranking(input_country, input_year, dataset):
    column=data_dictionary()
    country = column['Country by IMC']
    year = column['Year']
    #dataset=column[str(dataset)]
    count = []
    for x in range(len(year)):
        if input_year == year[x]:
            count.append(x)
    list = []
    for y in range(len(count)):
        if str(dataset[count[y]]) != 'n/a':
            list.append(float(dataset[count[y]]))
        else:
            list.append(float(-1000))
    country_list = country[count[0]:count[len(count)-1]+1]
    rank=len(list)+1-rankdata(list,method='max').astype(int)
    for x in range(len(count)):
        if input_country == country_list[x]:
            return rank[x]
```

Result

```
>>> column=data_dictionary()
>>>population = column["Population"]
>>>print str(ranking("Austria", "2009", population )) + "/190"
88/190
```

Build a country profile class

Script

```
class countryprofile(object):
    __country_classes = countrymap()
    def __init__(self, country, capital, continent, area, rank_area, year, gdppercapita,
rank_gdppercapita, unemploymentrate, rank_unemploymentrate, population, rank_population, debt,
rank_debt):
        """
            This class instance is created by country, capital, continent, area, rank_area, year,
gdppercapita, rank_gdppercapita, unemploymentrate, rank_unemploymentrate, population,
rank_population, debt, rank_debt.
            :param country:    name of the country
            :type country: str
            :param capital: capital of the country
            :type capital:    str
```

```
            :param continent:: continent of the country
            :type continent: str
            :param area: geographical area of the country
            :type area: int
            :param rank_area: rank of the geographical area among the country
            :type rank_area: int
            :param year: year of the macroeconomic data
            :type year: str
            :param gdppercapita: gdp per capita of the country
            :type gdppercapita: float
            :param rank_gdppercapita: rank of the gdp per capita among the country
            :type rank_gdppercapita: int
            :param unemploymentrate: unemployment rate of the country
            :type unemploymentrate: float
            :param rank_unemploymentrate: rank of the unemploymentrate among the country
            :type rank_unemploymentrate: int
            :param population: population of the country
            :type population: int
            :param rank_population: rank of the population among the country
            :type rank_population: int
            :param debt: government debt per GDP of the country
            :type debt: float
            :param rank_debt: rank of the government debt per GDP among the country
            :type rank_debt: int
            """
        self.__country=country
        self.__capital=capital
        self.__continent = continent
        self.__area = area
        self.__rank_area=rank_area
        self.__year = year
        self.__gdppercapita=gdppercapita
        self.__rank_gdppercapita = rank_gdppercapita
        self.__unemploymentrate=unemploymentrate
        self.__rank_unemploymentrate = rank_unemploymentrate
        self.__population=population
        self.__rank_population = rank_population
        self.__debt=debt
        self.__rank_debt = rank_debt

'''
Implement the getter methods
'''

@property
def country(self):
    return self.__country

@property
def capital(self):
    return self.__capital

@property
def continent(self):
    return self.__continent

@property
def area(self):
    return self.__area

@property
def rank_area(self):
    return self.__rank_area
```

```
@property
def year(self):
    return self.__year

@property
def gdppercapita(self):
    return self.__gdppercapita

@property
def rank_gdppercapita(self):
    return self.__rank_gdppercapita

@property
def unemploymentrate(self):
    return self.__unemploymentrate

@property
def rank_unemploymentrate(self):
    return self.__rank_unemploymentrate

@property
def population(self):
    return self.__population

@property
def rank_population(self):
    return self.__rank_population

@property
def debt(self):
    return self.__debt

@property
def rank_debt(self):
    return self.__rank_debt

'''
Implement the setter methods
'''

@country.setter
def country(self, name):
    self.__country = name

@capital.setter
def capital(self, name):
    self.__capital = name

@continent.setter
def continent(self, name):
    self.__continent = name

@area.setter
def area(self, value):
    self.__area = value

@rank_area.setter
def rank_area(self, value):
    self.__rank_area = value

@year.setter
def year(self, name):
```

```
        self.__year=name

    @gdppercapita.setter
    def gdppercapita(self, value):
        self.__gdppercapita = value

    @rank_gdppercapita.setter
    def rank_gdppercapita(self, value):
        self.__rank_gdppercapita = value

    @unemploymentrate.setter
    def unemploymentrate(self, value):
        self.__unemploymentrate = value

    @rank_unemploymentrate.setter
    def rank_unemploymentrate(self, value):
        self.__rank_unemploymentrate = value

    @population.setter
    def population(self, value):
        self.__population = value

    @rank_population.setter
    def rank_population(self, value):
        self.__rank_population = value

    @debt.setter
    def debt(self, value):
        self.__debt = value

    @rank_debt.setter
    def rank_debt(self, value):
        self.__rank_debt = value

    '''
        Implement the deleter methods
        '''

    @country.deleter
    def country(self):
        self.__country = ""

    @capital.deleter
    def capital(self):
        self.__capital = ""

    @continent.deleter
    def continent(self):
        self.__continent = ""

    @area.deleter
    def area(self):
        self.__area = 0

    @rank_area.deleter
    def rank_area(self):
        self.__rank_area = 0

    @year.deleter
    def year(self):
        self.__year = ""

    @gdppercapita.deleter
```

```
    def gdppercapita(self):
        self.__gdppercapita = 0

    @rank_gdppercapita.deleter
    def rank_gdppercapita(self):
        self.__rank_gdppercapita = 0

    @unemploymentrate.deleter
    def unemploymentrate(self):
        self.__unemploymentrate = 0

    @rank_unemploymentrate.deleter
    def rank_unemploymentrate(self):
        self.__rank_unemploymentrate = 0

    @population.deleter
    def population(self):
        self.__population = 0

    @rank_population.deleter
    def rank_population(self):
        self.__rank_population = 0

    @debt.deleter
    def debt(self):
        self.__debt = 0

    @rank_debt.deleter
    def rank_debt(self):
        self.__rank_debt = 0


    def __str__(self):
        """
        This function overwrites the built-in __str__ function to give a nice string representation of the
class content
        :return: string representation of the class content
        :rtype: str
        """
        return 'Country: {0}\nCapital: {1}\nContinent: {2}\nArea: {3} km2    ({4}/190)\nYear:
{5}\nGDP per capita: {6} U.S.dollars    ({7}/190)\nUnemployment Rate: {8}%    ({9}/190)\nPopulation:
{10} Millions    ({11}/190)\nGovernment Debt: {12} %GDP ({13}/190)'.format(
                str(self.country),
                str(self.capital),
                str(self.continent),
                str(self.area),
                str(self.rank_area),
                str(self.year),
                str(self.gdppercapita),
                str(self.rank_gdppercapita),
                str(self.unemploymentrate),
                str(self.rank_unemploymentrate),
                str(self.population),
                str(self.rank_population),
                str(self.debt),
                str(self.rank_debt))
```

Display the country profile and the ranking among the countries

Script

```
if __name__ == "__main__":
    column = data_dictionary()
```

```
        country = column['Country']
        capital = column['Capital']
        continent=column['Continent']
        area = column['Area']
        year = column['Year']
        gdppercapita = column['GDP percapita']
        unemploymentrate = column["Unemployment Rate"]
        population = column["Population"]
        debt = column["Government Net Debt"]
        while True:
            input1 = raw_input("Please enter a country(Uppercase matters!): ")
            if input1 in country:
                break
        while True:
            input2 = raw_input("Please enter a year(2006~2015): ")
            if input2 in year:
                break
        for x in range(len(year)):
            if input2 == year[x]:
                if input1 == country[x]:
                    information = countryprofile(country[x], capital[x], continent[x], area[x],
ranking(input1,input2,area), year[x],gdppercapita[x], ranking(input1,input2,gdppercapita),
unemploymentrate[x], ranking(input1,input2,unemploymentrate),population[x],
ranking(input1,input2,population),debt[x], ranking(input1,input2,debt))
                    print information
```

Result

```
>>> Please enter a country(Uppercase matters!):
Germany
>>> Please enter a year(2006~2015):
2008

Country: Germany
Capital: Berlin
Continent: Europe
Area: 357021 km2    (61/190)
Year: 2008
GDP per capita: 45976.12 U.S.dollars    (18/190)
Unemployment Rate: 7.41%    (44/190)
Population: 82 Millions    (14/190)
Government Debt: 47.955 %GDP (25/190)
```

(2) GeoMap.py

Import the packages we need and set the global variables

Script

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
from PySide.QtSvg import *
from PySide.QtGui import *
import os
import pylab
import shutil
# Map colors (http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=6)
colors={}
colors["GDP percapita"]=['#e31a1c', '#1f78b4', '#33a02c', '#b2df8a', '#fb9a99', '#a6cee3', '#ffff99',
'#CCCCCC']
colors["Unemployment Rate"]= ['#800026', '#e31a1c', '#fc4e2a', '#fd8d3c', '#fed976', '#ffeda0',
'#ffffcc', '#CCCCCC']
colors["Population"]= ['#000000', '#081d58', '#225ea8', '#1d91c0', '#41b6c4', '#c7e9b4', '#ffffd9',
'#CCCCCC']
colors["Government Net Debt"]=['#67000d', '#cb181d', '#fb6a4a', '#fcbba1', '#fee0d2', '#e5f5e0',
```

```
'#a1d99b', '#41ab5d', '#006d2c', '#00441b', '#CCCCCC']
threshold={}
threshold["GDP percapita"]=[30000,20000,10000,5000,3000,1000,0,"No data"]
threshold["Unemployment Rate"]=[15,13,10,7.5,5.5,4.5,0,"No data"]
threshold["Population"]=[1000,200,100,75,50,25,0,"No data"]
threshold["Government Net Debt"]=[100,87.5,75,62.5,50,37.5,25,12.5,0,-700,"No data"]
```

Build our dataset dictionary

Script

```
def data_dictionary():
    reader = csv.reader(open('worlddata.csv'), delimiter=",")
    #skip header and convert data to map
    headers = reader.next()
    column = {}
    for h in headers:
        column[h] = []
    for row in reader:
        for h, v in zip(headers, row):
            column[h].append(v)
    return column
```

Result

```
>>> print sorted(data_dictionary().keys())

['Alpha-3 code', 'Area', 'Capital', 'Continent', 'Country', 'Country Code', 'Country by IMC', 'GDP

percapita', 'Government Net Debt', 'Population', 'Telephone Code', 'Unemployment Rate', 'Year']
```

Build 2 methods for 2 inputs: indicator and year

Script

```
def input1():
    column = data_dictionary()
    indicator_list = ["GDP percapita", "Unemployment Rate", "Population", "Government Net Debt"]
    print "Indicators: " + str(indicator_list)
    while True:
        input1 = raw_input("Please enter an indicator you want to show on the world map: ")
        if input1 in column.keys():
            break
    return input1
def input2():
    column = data_dictionary()
    year = column['Year']
    while True:
        input2 = raw_input("Please enter a year(2006~2015): ")
        if input2 in year:
            break
    return input2
```

Result

```
>>> input1=input1()

Indicators: ['GDP percapita', 'Unemployment Rate', 'Population', 'Government Net Debt']

>>> Please enter an indicator you want to show on the world map:

Unemployment Rate

>>>input2=input2()

>>>Please enter a year(2006~2015):

2006
```

Edit the world map and save it to .txt format

Script

```
def worldmap():
    column = data_dictionary()
    countrycode = column["Country Code"]
    year = column['Year']
    count=[]
    for x in range(len(year)):
        if input2==year[x]:
            count.append(x)
    list = []
    for y in range(len(count)):
        if str(column[str(input1)][count[y]])!='n/a':
            list.append(float(column[str(input1)][count[y]]))
        else:
            list.append(float(-1000))
    my_xticks = countrycode[count[0]:count[len(count) -1]+1]
    indicator = {}
    for x in range(len(my_xticks)):
        try:
            country_id = my_xticks[x]
            index_value=list[x]
            indicator[country_id] = float(index_value)
        except:
            pass
    country=indicator.keys()
    # Load the SVG map
    svg = open('worldmap_revise.svg', 'r').read()
    # Load into Beautiful Soup
    soup = BeautifulSoup(svg, selfClosingTags=['defs','sodipodi:namedview'])

    if input1=="GDP percapita":
        # Find countries
        paths = soup.findAll('path')
        # Build the path style
        path_style =
'font-size:12px;fill-rule:nonzero;stroke:#000000;stroke-opacity:1;stroke-width:0.1;stroke-miterlimit:4;
stroke-dasharray:none;stroke-linecap:butt;marker-start:none;stroke-linejoin:bevel;fill:'
        # Color the counties based on GDP percapita
        country_list=[]
        for p in paths:
            country_list.append(p['id'])
            if p['id'] in country:
                threshold=indicator[p['id']]
                if threshold > 30000:
                    color_class = 0
                elif threshold > 20000:
                    color_class = 1
                elif threshold > 10000:
                    color_class = 2
                elif threshold > 5000:
                    color_class = 3
                elif threshold > 3000:
                    color_class = 4
                elif threshold > 1000:
                    color_class = 5
                elif threshold > 0:
                    color_class = 6
                else:
                    color_class = 7
                color = colors["GDP percapita"][color_class]
                p['style'] = path_style + color
    elif input1=="Unemployment Rate":
        paths = soup.findAll('path')
        path_style =
```

```
'font-size:12px;fill-rule:nonzero;stroke:#000000;stroke-opacity:1;stroke-width:0.1;stroke-miterlimit:4;
stroke-dasharray:none;stroke-linecap:butt;marker-start:none;stroke-linejoin:bevel;fill:'
            # Color the counties based on Unemployment Rate
            country_list = []
            for p in paths:
                country_list.append(p['id'])
                if p['id'] in country:
                    threshold = indicator[p['id']]
                    if threshold > 15:
                        color_class = 0
                    elif threshold > 13:
                        color_class = 1
                    elif threshold > 10:
                        color_class = 2
                    elif threshold > 7.5:
                        color_class = 3
                    elif threshold > 5.5:
                        color_class = 4
                    elif threshold > 4.5:
                        color_class = 5
                    elif threshold > 0:
                        color_class = 6
                    else:
                        color_class = 7
                    color = colors["Unemployment Rate"][color_class]
                    p['style'] = path_style + color
        elif input1=="Population":
            paths = soup.findAll('path')
            path_style =
'font-size:12px;fill-rule:nonzero;stroke:#000000;stroke-opacity:1;stroke-width:0.1;stroke-miterlimit:4;
stroke-dasharray:none;stroke-linecap:butt;marker-start:none;stroke-linejoin:bevel;fill:'
            # Color the counties based on Population
            country_list = []
            for p in paths:
                country_list.append(p['id'])
                if p['id'] in country:
                    threshold = indicator[p['id']]
                    if threshold > 1000:
                        color_class = 0
                    elif threshold > 200:
                        color_class = 1
                    elif threshold > 100:
                        color_class = 2
                    elif threshold > 75:
                        color_class = 3
                    elif threshold > 50:
                        color_class = 4
                    elif threshold > 25:
                        color_class = 5
                    elif threshold > 0:
                        color_class = 6
                    else:
                        color_class = 7
                    color = colors["Population"][color_class]
                    p['style'] = path_style + color
        elif input1=="Government Net Debt":
            paths = soup.findAll('path')
            path_style =
'font-size:12px;fill-rule:nonzero;stroke:#000000;stroke-opacity:1;stroke-width:0.1;stroke-miterlimit:4;
stroke-dasharray:none;stroke-linecap:butt;marker-start:none;stroke-linejoin:bevel;fill:'
            # Color the counties based on Government Net Debt
            country_list = []
            for p in paths:
```

```
                country_list.append(p['id'])
                if p['id'] in country:
                    threshold = indicator[p['id']]
                    if threshold > 100:
                        color_class = 0
                    elif threshold > 87.5:
                        color_class = 1
                    elif threshold > 75:
                        color_class = 2
                    elif threshold > 62.5:
                        color_class = 3
                    elif threshold > 50:
                        color_class = 4
                    elif threshold > 37.5:
                        color_class = 5
                    elif threshold > 25:
                        color_class = 6
                    elif threshold > 12.5:
                        color_class = 7
                    elif threshold > 0:
                        color_class = 8
                    elif threshold > -700:
                        color_class = 9
                    else:
                        color_class = 10
                    color = colors["Government Net Debt"][color_class]
                    p['style'] = path_style + color
        # Save the map format in .txt file
        file_index_txt = open('%s_%s.txt' % (input1, input2), 'w')
        file_index_txt.write(soup.prettify().encode('UTF-8'))
        # If wanting to generate an output as a svg file, you need to do some tricks as follow:
        # file_index_svg = open('%s_%s.svg' % (input1, input2), 'w')
        # file_index_svg.write(soup.prettify().encode('UTF-8'))
        # add text in line3 after <svg baseprofile="tiny" height="599px" viewbox="0 0 1360 599"
width="1360px" x="0px" y="0px"
```

Result `>>> worldmap()`

Convert .txt file to .jpg image

Script

```
def convert2jpg():
    #Revise the text file, in order to delete the <html>, <body>,</html>, and </body> in order for the
QSvgRenderer to read only the text between<svg>to </svg>
    f = open('%s_%s.txt' % (input1, input2), 'r+')
    d = f.readlines()
    f.seek(0)
    for i in d:
        if i.strip() != "<html>" and i.strip() != "<body>" and i.strip() != "</html>" and i.strip() !=
"</body>":
            f.write(i)
    f.truncate()
    f.close()
    #Open 2 empty image files: jpg and png, whcih can be edited
    pic_png = open("%s_%s.png" % (input1, input2), 'w')
    pic_jpg = open("%s_%s.jpg" % (input1, input2), 'w')
    #Put in the revise text file in QSvgRenderer function from PySide
    r = QSvgRenderer('%s_%s.txt' % (input1, input2))
    #Set the format of the image and save it as a png file
    height = r.defaultSize().height() * 950 / r.defaultSize().width()
    i = QImage(950, height, QImage.Format_ARGB32)
```

```
        p = QPainter(i)
        r.render(p)
        i.save("%s_%s.png" % (input1, input2))
        p.end()
        #Reopen the png image
        im = Image.open("%s_%s.png" % (input1, input2))
        # Open a new image, which is the same size as the png image we just saved
        # And set the new image background color: white
        bg = Image.new("RGB", im.size, (255, 255, 255))
        bg.paste(im, im)
        #We past our png image onto the new image and save it as a jpg file.
        bg.save("%s_%s.jpg" % (input1, input2))
        pic_jpg.close()
        pic_png.close()
        #Delete the png file which we're not going to use anymore
        os.remove("%s_%s.png" % (input1, input2))
```

Result >>> convert2jpg()

Build the legend.jpg image

Script

```
def legend():
    fig = plt.figure()
    ax    = fig.add_subplot(111)
    ax.set_axis_off()    #turn off the axis
    rate_patch = []
    for k in range(len(colors[input1])):
        if threshold[input1][k]!="No data":
            rate_patch.append(mpatches.Patch(color=colors[input1][k], label='>%s'
%threshold[input1][k]))
        else:
            rate_patch.append(mpatches.Patch(color=colors[input1][k], label='No data'))
    ax.legend(handles=rate_patch[0:],loc="lower right")
    fig.savefig('legend.png')
    # create legend as an image
    img = Image.open("legend.png")
    width = img.size[0]
    height = img.size[1]
    # .crop((left, top, right, bottom)
    img2 = img.crop(
        (
            width - 300,
            height - 420, #up
            width - 80,
            height - 30 #down
        )
    )
    # Instruction of image processing (https://yungyuc.github.io/oldtech/python/python_imaging.html)
    img2.save("legendsize.jpg")
    # paste image onto the map
    til = Image.open(("%s_%s.jpg" % (input1, input2)))    # 950x620
    im = Image.open("legendsize.jpg")    # 170x200
    width = 120
    ratio = float(width) / im.size[0]
    height = int(im.size[1] * ratio)
    nim = im.resize((width, height), Image.BILINEAR)
    til.paste(nim, (20, 400))
    til.save("legend.png")
```

Result >>> legend()

Complete the map: add a title and mix 3 elements (world map, legend, title)

Script

```
def completemap():
    # Add title
    newfile = Image.new("RGB", (1030, 700), color="#000000")
    newfile.save("world_%s_%s.jpg"% (input1, input2))
    image = Image.open("legend.png")
    newfile.paste(image, (40, 60))
    newfile.save("world_%s_%s.jpg"% (input1, input2))
    font = ImageFont.truetype(".../font/arial.ttf", 30)
    unit={"GDP percapita":"U.S.dollars","Unemployment
Rate":"%","Population":"Millions","Government Net Debt":"%GDP"}
    title = "World %s in %s (%s)" %(input1, input2,unit[input1])
    w, h = font.getsize(title)
    draw = ImageDraw.Draw(newfile)
    draw.text(((1030 - w) / 2, 10), title, color="white", font=font)
    newfile.save("world_%s_%s.jpg"% (input1, input2))
    # Create a new folder, and save the result into the folder
    newpath = r'./result'
    if not os.path.exists(newpath):
        os.makedirs(newpath)
    shutil.move("world_%s_%s.jpg" % (input1, input2), "result/world_%s_%s.jpg" % (input1, input2))
    shutil.move('%s_%s.txt' % (input1, input2), "result/%s_%s.txt" % (input1, input2))
    # remove the image that we don't need anymore
    os.remove("legendsize.jpg")
    os.remove("legend.png")
    os.remove("%s_%s.jpg" % (input1, input2))
```

Result

```
>>> completemap()
```

(3) Country Trends and Comparison.py

Import packages and read needed data files

Script

```
import numpy as np
import csv
import pandas as pd
```

```
import matplotlib.pyplot as plt
import shutil
import os
from datetime import datetime

# Read all the needed data files.
data1 = pd.read_csv("GDP.csv")
data2 = pd.read_csv("CPI.csv")
data3 = pd.read_csv("Net_debt.csv")
data4 = pd.read_csv("population.csv")
data5 = pd.read_csv("employment.csv")
data6 = pd.read_csv("unemployment_rate.csv")
```

Result >>>

Define country trend function

Script

```
def country_trend():
    # Let the user input the name of the country.
    input_country = raw_input("Please enter a country you want to know about:")

    # Give out the list of data type.
    print "-" * 25
    print "Data Type list:"
    print "1. GDP\n2. CPI\n3. Net_debt\n4. population\n5. employment\n6. unemployment_rate\n"
    input_type = raw_input("Please choose a type of trend you want to know from the list:")
    print "-" * 25
```

Result

```
>>>Please enter a country you want to know about:China
-------------------------
Data Type list:
1. GDP
2. CPI
3. Net_debt
4. population
5. employment
6. unemployment_rate

Please choose a type of trend you want to know from the list:CPI
-------------------------
```

Define single trend function (In the country trend function)

Script

```
def single_trend(data):
    # Show the first 10 rows of the chosen data type.
    input_country = raw_input("Please enter a country you want to know about:")

    print "The first 10 rows of data in the database"
    print data[['DATE', input_country]][:10]

    # Create a figure
    fig = plt.figure()

    # Create a subplot
    ax = fig.add_subplot(1, 1, 1)

    # Get the data to be drawn
    px = data[input_country]

    # Set the plot style
    px.plot(ax=ax, style='k-')
```

```
        # Set the title of the figure
        title = str(input_type + " of " + input_country)
        ax.set_title(title)

        # Set x-label for the figure
        ax.set_xlabel('Date')

        # Set y-label for the figure
        ax.set_ylabel(input_type)

        ax.set_xticklabels(range(1980, 2016, 4), rotation=30, fontsize='small')

        # Judge the data type and decide the corresponding file name for the figure
        if input_type == "GDP":
            save_name = str(input_country + " GDP.png")
        elif input_type == "CPI":
            save_name = str(input_country + " CPI.png")
        elif input_type == "Net_debt":
            save_name = str(input_country + " Net_debt.png")
        elif input_type == "population":
            save_name = str(input_country + " Population.png")
        elif input_type == "employment":
            save_name = str(input_country + " Employment.png")
        elif input_type == "unemployment_rate":
            save_name = str(input_country + " Unemployment_rate.png")

         # Save the figure to the right place, in the wanted style
         plt.savefig(save_name, dpi=400, bbox_inches='tight')

         # Open result folder if it's not exist    AND move the result to result folder
         newpath = r'./result'
         if not os.path.exists(newpath):
             os.makedirs(newpath)
        shutil.move('%s %s.png' % (input_country, input_type), "result/%s %s.png" % (input_country,
input_type))

        # Show and close the figure
        plt.show()
        plt.close()
```

Result

```
>>>prepare for figure, wait for the next function
The first 10 rows of data in the database
   DATE    China
0  1980   20.418
1  1981   20.929
2  1982   21.347
3  1983   21.774
4  1984   22.362
5  1985   24.442
6  1986   26.031
7  1987   27.931
8  1988   33.182
9  1989   39.155
```

Define input data type judging function (In the country trend function)

Script

```
# This function judges the input data type and returns different figure result.
def judge(input_type):
    if input_type == "GDP":
```

```
        data = data1
        single_trend(data)
    elif input_type == "CPI":
        data = data2
        single_trend(data)
    elif input_type == "Net_debt":
        data = data3
        single_trend(data)
    elif input_type == "population":
        data = data4
        single_trend(data)
    elif input_type == "employment":
        data = data5
        single_trend(data)
    elif input_type == "unemployment_rate":
        data = data6
        single_trend(data)
```

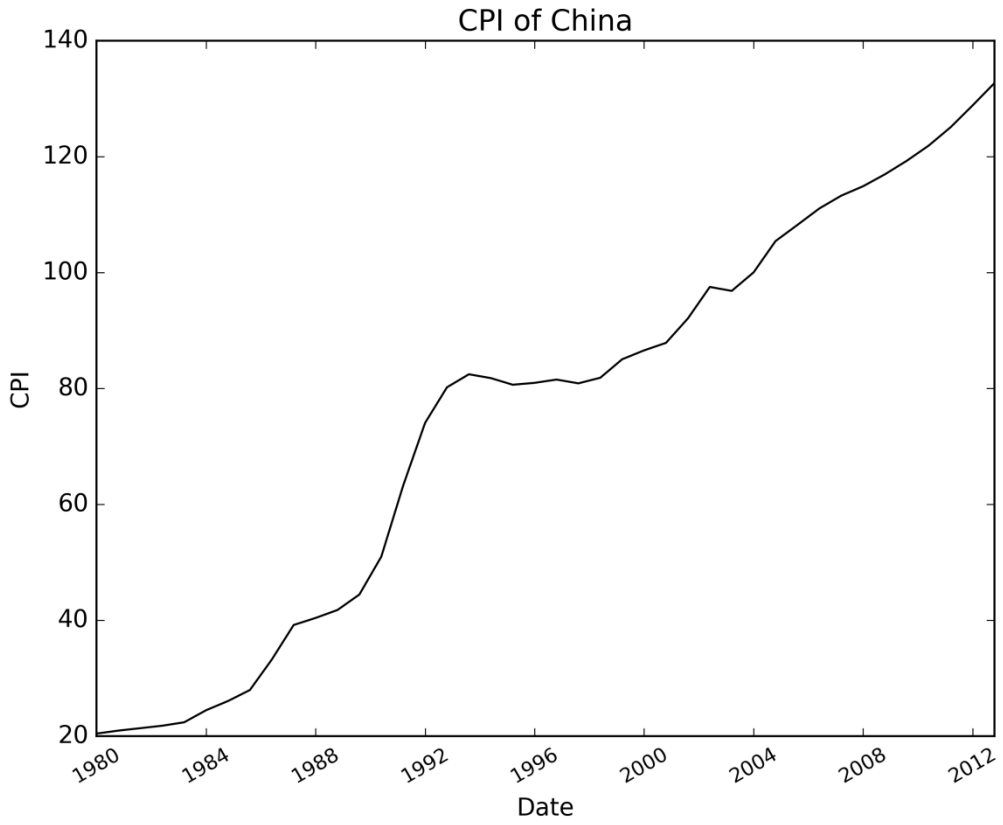
Result >>>judge(input_type)

Call the country trend function

Script

```
#Call the country_trend function
country_trend()
```

Result >>>country_trend()



Define comparison function

Script

```
# This function returns a figure comparing several countries' situations chosen by the user
def comparison():
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
```

```
        # Get the chosen type of data
        print "-"*25
        print "Data Type list:"
        print "1. GDP\n2. CPI\n3. Net_debt\n4. population\n5. employment\n6. unemployment_rate\n"
        print "-"*25

        chosen_type = raw_input("Please enter the type of data you want to compare:")

        if chosen_type == "GDP":
                data = data1
        elif chosen_type == "CPI":
                data = data2
        elif chosen_type == "Net_debt":
                data = data3
        elif chosen_type == "population":
                data = data4
        elif chosen_type == "employment":
                data = data5
        elif chosen_type == "unemployment_rate":
                data = data6
        else:
                chosen_type = raw_input("Sorry, but the type you put in is not in our database, please
re-enter a type.")

        # Get the chosen countries
        chosen_countries = raw_input("Please enter several countries you want to compare, split by
','/like 'China,United States'. Please do not input blanks, Upper case counts!:")

        # split the string user inputs
        chosen = chosen_countries.split(",")

        # Create a list for the maximum and minimum numbers of each country's data
        y_maxes = []
        y_mins = []

        # Print the chosen countries' situations and get the upper limit and lower limit for the figure
        for country in chosen:
                print "Last ten years'",chosen_type,"of",country
                print "*" * 25
                print data[['DATE', country]][-10:]
                print "*" * 25

                # put the maximum and minimum of each country's data into the original list
                y_maxes.append(max(data[country]))
                y_mins.append(min(data[country]))

                # Draw each country's line
                ax.plot(data[country], 'o-', label = country)

        # Get the maximum among all the country's max and decide the upper limit
        max_y = str(max(y_maxes))
        len_max = 0
        for num in max_y:
                if num.isdigit() is True:
                        len_max += 1
                else:
                        break
        upp_lim = (int(max_y[0])+1) * 10 ** (len_max - 1)

        # Get the minimum among all the country's min and decide the lower limit
        min_y = str(min(y_mins))
        len_min = 0
        for num in min_y:
```

```
            if num.isdigit() is True:
                len_min += 1
            else:
                break
        low_lim = int(max_y[0]) * 10 ** (len_min - 1)

        # Set the title of the figure
        ax.set_title('Country Comparison')

        # Set the y label of the figure
        ax.set_ylabel(chosen_type)

        # Set the x label of the figure
        ax.set_xlabel('DATE')

        # Set the x tick labels as every year
        ax.set_xticklabels(range(1980, 2016, 4), rotation = 30, fontsize = 'small')

        # Set the y limits with low_lim and upp_lim we get before
        ax.set_ylim(low_lim, upp_lim)

        # Put the legend at the best place
        plt.legend(loc = 'best')

        # Judge the data type and decide the corresponding file name for the figure
        if chosen_type == "GDP":
            save_name = "GDP Comparison.jpg"
        elif chosen_type == "CPI":
            save_name = "CPI Comparison.jpg"
        elif chosen_type == "Net_debt":
            save_name = "Net Debt Comparison.jpg"
        elif chosen_type == "population":
            save_name = "Population Comparison.jpg"
        elif chosen_type == "employment":
            save_name = "Employment Comparison.jpg"
        elif chosen_type == "unemployment_rate":
            save_name = "Unemployment Rate Comparison.jpg"

        plt.show()
        plt.savefig(save_name)
```

Result

```
>>>comparison()
-------------------------
Data Type list:
1. GDP
2. CPI
3. Net_debt
4. population
5. employment
6. unemployment_rate

-------------------------
Please enter the type of data you want to compare:GDP
Please enter several countries you want to compare, split by ','/like 'China,United States'. Please do not input blanks, Upper case counts!:China,United States,United Kingdom,Germany
Last ten years' GDP of China
*************************
    DATE      China
26  2006    5787.80
27  2007    6750.57
28  2008    7505.53
29  2009    8218.18
```

30	2010	9156.88
31	2011	10180.86
32	2012	11111.60
33	2013	12102.56
34	2014	13130.87
35	2015	14107.43

****

Last ten years' GDP of United States

****

	DATE	United States
26	2006	46351.67
27	2007	47954.53
28	2008	48302.28
29	2009	46909.42
30	2010	48309.53
31	2011	49725.50
32	2012	51385.49
33	2013	52615.34
34	2014	54360.50
35	2015	55805.20

****

Last ten years' GDP of United Kingdom

****

	DATE	United Kingdom
26	2006	34690.78
27	2007	36294.07
28	2008	36586.09
29	2009	35093.41
30	2010	35797.00
31	2011	36654.34
32	2012	37520.68
33	2013	38723.66
34	2014	40163.30
35	2015	41158.91

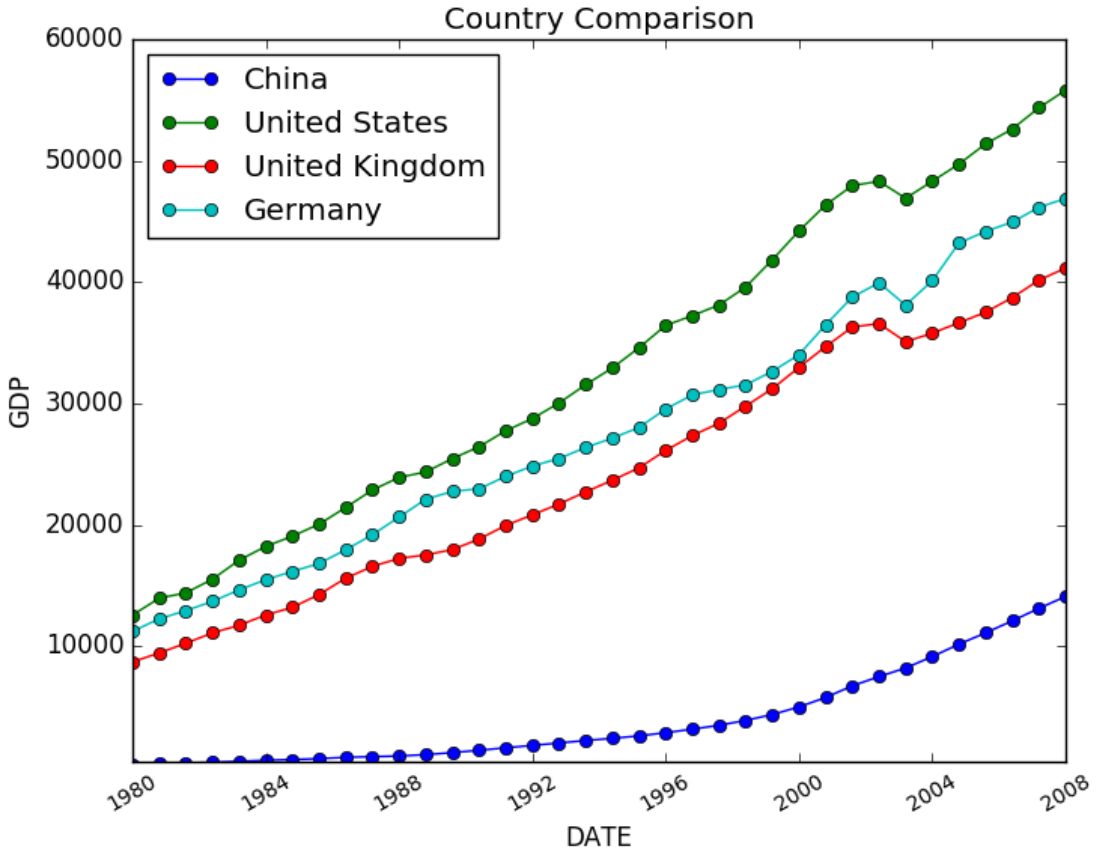
****

Last ten years' GDP of Germany

****

	DATE	Germany
26	2006	36482.13
27	2007	38765.71
28	2008	39949.20
29	2009	38105.39
30	2010	40117.59
31	2011	43220.71
32	2012	44179.10
33	2013	44945.64
34	2014	46160.19
35	2015	46893.17

****



(4) Convert Image to Gif.py

Import needed packages function

Script

```
import os
from PIL import Image
import images2gif
```

Result

```
>>>
```

Define convert_to_gif function function

Script

```
def convert_to_gif(target_gif_Path, image_file_paths, type = 0):
    # Get the images needed to convert to gif.
    images = []

    # Get the max length among the images
    max_width_and_height = 1

    # Max width and height
    max_width = 1
    max_height = 1

    # Sort the images by width
    width_and_file_paths = []

    # Sort the images by height
    height_and_file_paths = []

    # Open the images and get related information
    for image_path in image_file_paths:
        fp = open(image_path, "rb")
```

```
            width, height = Image.open(fp).size
            width_and_file_paths.append((width, image_path))
            height_and_file_paths.append((height, image_path))
            max_width = max(max_width, width)
            max_height = max(max_height, height)
            fp.close()

        # Get the max width and height
        max_width_and_height = max(max_width_and_height, max_width, max_height)

        # Sort the width and height in descending order
        width_and_file_paths.sort(key=lambda item: item[0], reverse=True)
        height_and_file_paths.sort(key=lambda item: item[0], reverse=True)

        # Choose the style to sort the images
        if type == 4 or type == 5:
            # Convert the original figures directly ordered by width
            if type == 4:
                for width_and_file_path in width_and_file_paths:
                    img = Image.open(width_and_file_path[1])
                    images.append(img)
                    # Convert the original figures directly ordered by height
            if type == 5:
                for height_and_file_Path in height_and_file_paths:
                    img = Image.open(height_and_file_path[1])
                    images.append(img)

        else:
            for image_file_path in image_file_paths:
                fp = open(image_file_path, "rb")
                img = Image.open(fp)
                width, height = img.size
                # Build a white background canvas
                if type == 0 or type == 2:
                    # rectangular
                    imgResizeAndCenter = Image.new("RGB", [max_width, max_height], (255, 255,255))
                elif type == 1 or type == 3:
                    # quadrate
                    imgResizeAndCenter = Image.new("RGB", [max_width_and_height,
max_width_and_height], (255, 255, 255))

                if type == 0:
                    # max width >= max height, revise the size a little bit
                    if max_width / width >= max_height / height:
                        resizeImg = img.resize((width * max_height / height, max_height),
Image.ANTIALIAS)
                        imgResizeAndCenter.paste(resizeImg, ((max_width - width * max_height /
height) / 2, 0))
                    else:
                        resizeImg = img.resize((max_width, height * max_width / width),
Image.ANTIALIAS)
                        imgResizeAndCenter.paste(resizeImg, (0, (max_height - height * max_width /
width) / 2))

                if type == 1:
                    # width >= height, zoom the width to max length
                    if width >= height:
                        resizeImg = img.resize((max_width_and_height, height *
max_width_and_height / width), Image.ANTIALIAS)
                        imgResizeAndCenter.paste(resizeImg,
                                                 (0, (max_width_and_height - height *
max_width_and_height / width) / 2))
                    else:
```

```
                    resizeImg = img.resize((width * max_width_and_height / height,
max_width_and_height), Image.ANTIALIAS)
                    imgResizeAndCenter.paste(resizeImg,
                                             ((max_width_and_height - width *
max_width_and_height / height) / 2, 0))
                elif type == 2:
                    imgResizeAndCenter.paste(img, ((max_width - width) / 2, (max_height - height) / 2))
                elif type == 3:
                    imgResizeAndCenter.paste(img, ((max_width_and_height - width) / 2,
(max_width_and_height - height) / 2))

                #Save the images
                imgResizeAndCenter.convert("RGB").save(os.path.dirname(image_file_path) + os.sep +
"ResizeAndCenter" + os.path.basename(image_file_path), 'jpeg')
                images.append(imgResizeAndCenter)
                fp.close()
```

Result

```
>>>images2gif.writeGif(target_gif_Path, images, duration=1, nq=0.1)
```

Call the main function

Script

```
if __name__ == "__main__":
    # Open result folder if it's not exist
    newpath = r'./result'
    if not os.path.exists(newpath):
        os.makedirs(newpath)

    #Save the GIF file
    convert_to_gif(r"result\convert.gif",
                   [r"world_gdppercapita 2006-2015\world_GDP percapita_2006.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2007.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2008.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2009.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2010.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2011.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2012.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2013.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2014.jpg",
                    r"world_gdppercapita 2006-2015\world_GDP percapita_2015.jpg",],
                    2)
```

Result

```
>>> # The gif file does not suit dynamic presenting here. Please read it in Internet explorer.
```

4. Q&A

(1) What can I do if I want to insert new data?

Open the **worlddata.csv** and add a new column. Name your dataset in the first row and then add the data.

(2) What can I do if I want to build my own map?

You can freely download the map from Amcharts (https://www.amcharts.com/svg-maps/) and replace the **worldmap.svg** file into your map and create your own style. Make sure that the svg file's path fit the 'Country Code' column in **worlddata.csv**, otherwise the data cannot be read in python file.

(3) Can I change the time period? Monthly? Daily?

Yes. You need to change the value in the 'Year' column in **worlddata.csv** and set it to the time period you're going to choose.

(4) How can I open the gif file? What if I cannot see the animation effect?

Please open the gif file in the internet explorer so that it is easy for you to see the animation effect, changing in a frequency of once a 0.1 second. And if you cannot see the gif file normally, please adjust your Internet explorer's setting.

(5) What can I do if I have problems running the .py file or I have some suggestion after using the project?

Making improvement and working on a better coding structure is our goal. Please don't hesitate to contact us: C.C, Hung (lisa5432126@gmail.com) and X.Y, Lai (laixiaoya96@163.com). We will give you response as soon as possible.