Eberhard Karls Universität Tübingen
Economics and Business Administration(BA)
Introductory to Python
Spring Semester 2016
Marisa Köllner, Johannes Wahle

Macroeconomic Data Visualization with Python

Chia-Chein, Hung
National Tsing-Hua University (NTHU)
lisa5432126@gmail.com
Xiao-Ya, Lai
Renmin University of China (RUC)
laixiaoya96@163.com
24th September, 2016

Abstract:

In this essay, we elaborate the background and contents of our program, focusing on macroeconomic data visualization with Python statements and packages, explaining the situations and trends of each country's economy.

Keywords:

Macroeconomics, Data, Data Visualization, Python

1. Background

There exist quantities of soft-wares or programming languages facilitating researchers to process, integrate and then analyze the data they obtain from experiments or researches. Among all the powerful tools, Python is not an omnipotent one, but exhibits many distinctive advantages like its simplicity in coding, and the convenience in its good combination with other effective functional packages. Also, its abundant functions achieving plenty of mathematical calculations or other analysis are also appealing to those who are interested in data analyzing.

Consequently, after we failed in accomplishing feature analysis and author attribution of Shakespeare's workings, we considered fixing our attention on data visualization with Python. Besides, as we are both from financial related majors, using financial or economic data as our database will be a relatively easier way — as it will prevent us from wasting too much time in considering which kind of index to use, or the way we choose to process or analyze the data, and thus allow us to concentrate our time on coding and programming. Also, choosing economic and financial data to translate into graphics have more realistic significance, so that the results we get can be easily examined and the hidden information can also be more easily discovered.

To decide on the exact types of data from a mass amount of economic data is then the biggest problem we need to solve. We took a glance at several relative papers, and finally decided on using the IMF database, which contains a full set of economic data from both developed and developing countries and can enable us easily comparing the situation and future trends of these countries. Besides, with the help of a full database, we can get relatively more comprehensive graphs, covering wider range of information needed.

Considering we are using Python as a tool to visualize all the economic data, the things we focus on will be in two aspects. Firstly, after planning on our pseudocode, we need to simplify and purify our codes, making them more readable, reasonable and effective; the other one is that we need to make the results seem more understandable so that people with little economic knowledge can get easy access to what we have constructed and concluded, as well as make sense of what we want to present by our coding and our graph results.

After all these background consideration, using purified Python codes to visualize our selected economic data and thus show a understandable world economic situation and trend is what we want to accomplish in our program.

2. Introduction

(1) Definition

From Wikipedia, we got the definition of data visualization. Data visualization is viewed by many disciplines as a modern equivalent visual communication. It

involves the creation and study of visual representation of data, meaning "information that has been abstracted in some schematic form, including attributes or variables for the units of information."

A primary goal of data visualization is to communicate information clearly and efficiently via statistical graphs, plots and information graphics, making it convenient for readers or researchers to examine the hidden characteristics of those seemingly "superficial" data. On the meanwhile, data visualization also reveals the relationships or connections among those seemingly unrelated data.

The reason why we choose python to do data analysis and data visualization is not simply because that Python language is easy to fall in love with, compared to C++ and Java. Also Python is distinguished by its large and active scientific computing community. Adoption of Python for scientific computing in both industry applications and academic research has increased significantly since the early 2000s (Wes M., 2012).

For data analysis, interactive computing and data visualization, Python have various packages which can easily be compatible with other programming languages and tools in wide use, such as R and MATLAB. This certainly makes Python a strong alternative for data manipulation tasks. What's more, data visualization by Python requests us encoding all the needed numerical data by means of dots, lines or bars, with help of Python statements and packages, to visually communicate a quantitative message (Keven S., 2014).

(2) Goal

The initial goal of our coding is to find the most effective way to accomplish data visualization with Python, and make full use of all the packages we have learned and got hang of during these days. And thus improving our ability of effectively using Python is in later studies.

After fulfilling the process of Python coding, we will transfer the originally complex and tiring economic data to more accessible, usable and understandable ones, and realize our purpose of explaining the world economic situation and presenting world economic trend to those who have little economic knowledge.

(3) Method

We will firstly construct a database, containing the data of all the countries we get from the IMF data base, and if needed, users can access them from the class called "country profile". By inputting the name of the country a user wants to know, the database function will return all the information of the country, as well as the graphics of its GDP, unemployment rate, etc.

Then we construct another function putting all countries' information on a world map, and showing their differences on economic development or social

status by using different colors on the map, and also showing the disparity of their extent of suffer from financial crisis.

After getting the data-present-maps, we create a function (independently exist as an independent python file) to unify the maps into an integrated .gif file. This function allows us to show the maps containing data from different years dynamically, constructing more visually attracting results. And with this method of comparison, the differences among all the countries as well as the changes of one specific country will be more easily detectable.

The next step we design is to build a function returning the contrasting graphics among the user picked countries, visually vividly showing the differences of their economic situations by plots and histograms, according to their data saved in the database, and showing their own characteristics. Users can choose the countries and the type of data they want to compare from the given list, and then get the results of comparison in line(plot) graphs. These graphics will also be able to show several interesting differences of these countries' reactions to financial crisis, maybe some will be identified as smarter countries recovering rapidly from crisis and others as passive ones.

In the program, we also create a ranking function (included in country profile and the ranking.py) which put all the countries into a sequence by certain criteria, and then return the rank of the country user puts in to the function, and thus help the users know more about the country they put into the program.

The results of our program will be several graphics (visualization results) talking about the economic development features of all the countries we include in our database, and consequently exhibit what we learned in data visualization by Python, with all the functions we are exposed to throughout the semester, and the packages we knew and got hang of during the process of preparing for the project.

3. Toolbox

According to our project development, we use several useful packages and libraries to help us well-organized our dataset and bring prosperity to the function we are eager to fulfill.

(1) CSV

CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. There is no "CSV standard", so the format is operationally defined by many applications which read and write it. The csv module implements classes to read and write tabular data in CSV format. Therefore, we mainly use CSV dictionary for reading files.

(2) Numpy

NumPy (Numerical Python) is the foundational package for scientific computing in

Python. We use this for building a fast and efficient multidimensional array object ndarray. And also we use some linear algebra operations and random number generation. With regards to data analysis, Numpy is the primary container for data to be passed between algorithms. Also, for numerical data, NumPy arrays are a much more efficient way of storing and manipulating data than the other built-in Python data structures.

(3) Scipy

SciPy is a collection of packages addressing a number of different standard problem domains in scientific computing. Scipy contains a lot of different packages for different computing methods, for example, scipy.integrate, scipy.linalg, scipy.optimize, scipy.signal, scipy.stats, etc. We mainly use **scipy.stats** to generate standard continuous and discrete probability distributions, various statistical tests, and more descriptive statistics.

(4) Pandas

Pandas provides rich data structures and functions designed to make working with structured data fast, easy, and expressive. It is mostly used for creating a powerful data analysis environment. And for financial dataset, pandas features rich, high-performance time series functionality and tools well-suited for working with financial data (Wes, M. and PyData Development Team, 2016). We mainly use pandas for displaying the table in csv file, which shows a two-dimensional tabular, column-oriented data structure with both row and column labels.

(5) Matplotlib

Matplotlib is a useful Python library for producing plots and other 2D data visualizations. It was originally created by John D. Hunter (JDH) and is now maintained by a large team of developers. It is well-suited for creating plots suitable for publication. It provides a comfortable interactive environment for plotting and exploring data. We use Matplotlib mainly for plotting the statistics graphs, including time series graphs and histograms. Additionally Matplotlib integrated with IPython provides an interactive research and development environment with data visualization suitable for most users.

(6) Beautifulsoup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It provides idiomatic ways of navigating, searching, and modifying the parse tree. We use this tool to add information onto our world map document (which is a SVG[1] file related to HTML).

(7) Datetime

[1] SVG (Scalable Vector Graphics) is an XML file, which texts with tags, and can be edited in a text editor just like a HTML file. The browser or image viewer reads the XML and the XML tells the browser what to show.

Datetime is a Python package able to transfer string type to date type data, or vice versa. In this program, this package serves to process the date data contained in the csv file and plot them in the graphics.

(8) os

Os is a Python package providing a portable way of using operating system dependent functionality, especially give some systematical judgments to help decide the path, etc.

(9) Images2gif

Images2gif is a package to read and write animated gifs. We use the package to get animated effects out of the static geo-map graphics.

(10) PIL

The Python Imaging Library (PIL) adds image processing capabilities to the Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing and graphics capabilities. We use this library for transforming our image format and also for image processing, including image resizing, rotation and arbitrary affine transforms.

4. Dataset Interpretation

By analyzing the data, we found the open source on the Internet, the publication from International Monetary Fund (IMF). The publication is **The World Economic Outlook (WEO) database**, **April 2016**, which contains selected macroeconomic data series from the statistical appendix of the World Economic Outlook report, which presents the IMF staff's analysis and projections of economic developments at the global level, in major country groups and in many individual countries. The WEO is released twice a year, in April and September/October and data are available for 191 countries from 1980 to the present. However, some countries' data are incomplete or unavailable for certain years due to political or economic reasons.

We used this database to select data on macroeconomic indicators, including GDP per capita (U.S. dollars), unemployment rate (%), population (Millions), government debts (per GDP). And we selected the yearly dataset from 2006 to 2015 within 190 countries. The reason why we selected only 10 years was because that the period contained the **Financial Crisis of 2007–08**, which might be apparently to show on the plot and map by using data analysis and data visualization methods.

We also found the country profile's indicator from **Wikipedia** and **United Nations Statistics Division**, including country code (Alpha-2 code and Alpha-3 code[2]), capital, telephone code and geographical area (km^2).

[2] Alpha-2 code and Alpha-3 code are related to ISO 3166 standard, published by the International Organization for Standardization (ISO) They define codes for the names of countries, dependent territories, and special areas of geographical interest.

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 country), **Country Code** (Alpha-2 code), **Alpha-3 code**, **Capital**, **Continent** (categorized into 6 continents: Africa, Asia, Europe, North America, South America and Oceania), **Area** (km^2, 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 were collected), **GDP percapita** (U.S. dollars), **Unemployment Rate** (%), **Population** (Millions), **Government Net Debt** (per GDP).

The method we use for building the dataset is by pooling the time series data into one column. For example, the **Population** column contains the data among 190 countries in 10 years (from 2006 to 2015). Therefore the amount of data value in one column is 1900. By doing so, we can easily select data by its specific category.

For the purpose of deep and independent researching, we extract the time series data of all countries from the original data file and then put into one independent csv file, and draw figures separately from the independent files. Users can input a country they want to know about, and input the type of data (e.g. GDP), then they will get a line plot graph showing the trend of changing of the specific data in the particular country. And the additional six .csv files are all for this purpose, containing all the data from IMF dating back to 1980 and to 2021 (predictive), ranging from GDP to unemployment rate, comprehensively showing the economic features of the countries in the world.

5. Country Profile and the Ranking

(1) Script Interpretation

We first decide to display the country information and the ranking among countries. For the user who has less knowledge on economic interpretation, the better way to transmit the information is to use a simple input and the ranking method to build up the knowledge behind the numerical information. And also, in a huge dataset, it's often hard to find the information we necessarily require, therefore we try to simplify the process and to create a much easier user experience to generate useful information behind the data.

First, we import the packages we need, including ***csv, scipy.stats, pandas***. Then we set the format for displaying the table of our data file. This step is to let the user have the background knowledge of our dataset. Next, we build our dataset dictionary. Instead of the tabular form in csv file, we transform it into the dictionary. The keys as following: ***Country, Country Code, Alpha-3 code, Capital, Continent, Telephone Code, Area, Year, GDP percapita, Unemployment Rate, Population, Government Net Debt***, are the header of the csv file, which are the

titles of each dataset and the values are based on each title. Continually, we build a ranking method: ***ranking()*** by using ***rankdata*** in ***scipy.stats*** package and the method we choose is **max**, which means if there exists the same value of the dataset, then the maximum of the ranks that would have been assigned to all the tied values will be assigned to each value. Within the method: **ranking(*input_country, input_year, dataset*)**, by inputting (the country name, the Year, the title of the dataset we're going to find), it will generate the ranking of the dataset in the year where the country's position is. For example, by typing ***print str(ranking("Austria", "2009", population)) + "/190",*** the output will be ***88/190.*** This means that among 190 countries, the population of Austria in 2009 ranked as 88.

The final step is to build a country profile class. The fourteen variables as following: **country, capital, continent, area, rank_area, year, gdppercapita, rank_gdppercapita, unemploymentrate, rank_unemploymentrate, population, rank_population, debt, rank_debt**. And we implement the getter methods, the setter methods and the deleter methods of 14 variables. Also, we give a nice string representation of the class content by setting up a specific format.

(2) Results

After completing the building of the methods and class, we want to display the country profile and the ranking among the countries. The user simply needs to input 2 variables: **input1** and **input2**.

For input1, the user needs to enter a country name, which the uppercase matters, and for input2, the user needs to enter a year in the range of 2006 to 2015. After entering input1 and input2, which the format is correct, it will directly come out the result of the country's basic profile and the ranking of the macroeconomic indicators among 190 countries. For example, if we want to know the profile of Germany in 2008, the result will be as following:

>>> 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)

As you can see, the result shows that the capital and the continent of Germany are Berlin and Europe respectively. The geographical area ranks 61 among 190 countries. In 2008, the GDP per capita of Germany ranked rather high, which was the first 10% compared to the other countries. However, the unemployment rate was also high, which ranked 44 among 190 countries. By compiling this python file, you can easily see the countries' basic profile and the ranks of the macroeconomic indicator compared to the other countries. For the user who has less knowledge on economic interpretation, he can transmit the information by using 2 simple inputs and the ranking method to build up the knowledge behind the numerical information. And the user can also see the changing of the data value and ranking by entering different year, which shows the country's economy condition compared to the other countries.

6. GeoMap Visualization

We want to build a world map combining the macroeconomic data information for the user who doesn't understand the GIF method and lack of economy knowledge (Nathan Y., 2009). By our method, they can easily input the dataset and come out a beautiful GeoMap, which is based on the yearly dataset of the macroeconomic indicator.

(1) Script Interpretation

The first step of data visualization is to get the data. Therefore, we use the macroeconomic yearly indicators from 2006 to 2015 among 190 countries. The dataset is built by pooling the time series data into one data set. By doing so, we can easily select data by its specific category, including year, country, macroeconomic indicator, etc.

First, we import the packages we need, including ***csv, BeautifulSoup, Image, PIL, ImageFont, svgwrite, svgutils, string, matplotlib, numpy, PySide, os, pylab, shutil***. Then we set the format for the color set we are going to display on the world map by using **ColorBrewer**, and also the threshold for each color set. Because we knew the distribution of the indicator beforehand; therefore we set up a specific threshold for each color set. However, if not knowing the distribution, we could use something like this to build the threshold: **float(len(colors)-1) * float(indicator_value - min_value) / float(max_value - min_value)**.

After building the environment, we build our dataset dictionary. Instead of the tabular form in csv file, we transform it into the dictionary. The keys as following: ***Country, Country Code, Year, GDP percapita, Unemployment Rate, Population, Government Net Debt***, are the header of the csv file, which are the titles of each dataset and the values are based on each title.

Continually, the user needs to input 2 variables: **input1** and **input2**. For input1,

the user needs to enter an indicator that he wants to show on the world map. The indicators we choose are 4 macroeconomic indicators: ***GDP percapita, Unemployment Rate, Population, Government Net Debt***, which are the important indexes for national economy perspectives. For input2, the user needs to enter a year in the range of 2006 to 2015, in which the specific year to display the data on the world map. Then we started to build the world map.

First of all, we get a blank world map from **Amcharts (**Amcharts.com, 2006**)**, where we can freely download the world map in SVG format. As soon as we open the SVG file, we can see that each path is a county, which is written in the country code (Alpha-2 code). The country code is followed by a set of numbers, which are the coordinates for the country's boundary lines. We read in the data from the method **data_dictionary()** which we've built previously. And we write a loop to get the data from the specific year the user entered, which is actually the **input2**.

Continuously, we build an indicator dictionary: **indicator={}**. The **indicator value** in specific year is then stored in the **indicator dictionary** with the **Country code** as the key. Later on, we open the SVG world map and load SVG into **Beautiful Soup**. Beautiful Soup has a **findAll()** function that we can use to find all the counties in our SVG file by searching for the **path** in SVG file. Because we're interested in filling the color on each country, we change the style attribute for each path in the SVG and replace the entire style instead of parsing to replace only the color. Everything is the same as the previous style except we move **fill** to the end and left the value blank and we also changed stroke to **#000000** to make county borders black while fill depends on the indicator.

Depending on different indicators, we loop through all the paths to find the indicator value from the indicator dictionary, and select color from **colors{}** according to the indicator's threshold we've built previously **threshold["indicator"]**. Overall, we output the newly SVG colored map into txt file by encoding **soup.prettify()** with **'UTF-8'** in order to save it as an image file.

Next, we build a method: **convert2jpg()** in order to convert the text file to image file in jpg format. Pyside has a **QSvgRenderer()** function, which can render the texts into image. But, we need to first revise the text file by deleting the lines including **<html>, <body>,</html>, </body>,** in order for the **QSvgRenderer** to read only the **<svg>** texts. And we set the format of the image and save it as a png file. And by cutting the png file and pasting on a new white image, we resave it into jpg file.

After successfully converting the text file to image file. Now we have a beautiful colored world map related to the indicator. However, we're lack of the legend on the image. So we build a method: **legend()** to create a legend image. We

use the function in **matplotlib**. However, by only saving the legend image, we use **ax.set_axis_off()** to turn off the axis which automatically generate by plotting a graph in matplotlib. The legend includes the colorset of each threshold and the explanation of the threshold.

The last step is to complete the map by pasting the world map, the legend together and add a title on the image. The title shows like this: **"World *(Indicator)* in *(Year)* *(Indicator's unit)*"**. And by cutting, copying and pasting, we can successfully save our complete world map into jpg format: **"world_*(Indicator)*_*(Year)*.jpg"**. The final step is to move our output: text file and image file to the result folder by **shutil.move()** and delete the unusable image by **os.remove()**.

(2) Results

Finally, there's a result folder which saves the output by implementing the previously methods and shows the results of the world map with 2 simple inputs: **indicator**, **year**. We take **Unemployment Rate** as an example:

```
>>> Indicators: ['GDP percapita', 'Unemployment Rate', 'Population', 'Government Net Debt']
>>> Please enter an indicator you want to show on the world map:
Unemployment Rate
>>> Please enter a year(2006~2015):
2007
```

We select the indicator: **Unemployment Rate**, which we're interested in. The time period we choose was from 2007 to 2009, which had a strong correlation to **Financial Crisis of 2007–08**. And there come three maps: **World Unemployment Rate in 2007 (%), World Unemployment Rate in 2008 (%), and World Unemployment Rate in 2009 (%)**.

According the information shows on the map, we discover that the countries in South America and Africa had a great impact on suffering the higher unemployment rate compared to the other continents. The unemployment rate remained high and didn't decline from 2007 to 2009. While the other continents like Europe, North America, Asia and Oceania had a significant change from 2008 to 2009. The unemployment rate increased at least 3% within these countries.

To conclude, the map shows that within different countries, there might be a huge difference on the economic statement based on the different color set on the map. And for the user, he can easily discover the yearly change among countries and showed the relationship between them, despite the lack of economic knowledge.

Picture1. The Unemployment Rate (%) in different countries in 2007

Picture2. The Unemployment Rate (%) in different countries in 2008

Picture3. The Unemployment Rate (%) in different countries in 2009

7. Statistical country situation and comparison visualization

In this part, we want to visualize the country's macroeconomic situations statistically. As we are focusing on the trend visualization, the most frequent types of graph we are using here is line plots. With the data provided by the csv file and according to the country user chooses, the function will give out the wanted visualization result.

(1) Script Interpretation

The first step here, like what we do in GeoMap Visualization, is to get the data we need. We extract our needed data separately from the original IMF data, and our files covers macroeconomic data of all the countries from 1980 to 2015, with some predictions for 2016 to 2021.

Then we import the needed packages. In the country trend and comparison function, the packages we need are **numpy**, **csv**, **pandas**, **matplotlib** and **datetime**. Then we use the csv package to read the data we need. We prepared six files here, **GDP**, **CPI**, **population**, **employment**, **unemployment rate** and **net debt**. And we put the data of each file into data1, data2 to data6 and form six dataframes. After we save the data into the dataframes, we are able to extract a specific column or row from them.

After we arrange the data, we are able to process them now. To achieve country trend comparison, we define a function called **country_trend()**, in which there are two functions naming **single_trend(data)**, and **judge(input_type)** providing figuring and judgment effects.

In the first part of country trend function, we arrange a raw input function to get user's input objects (countries and types). After getting the user's intended countries and types, we use the **single_trend(data)** and **judge(input_type)** functions to draw the needed figures.

In **single_trend(data)** function, we firstly print out the first ten rows of the chosen data, and then use the statement **fig = plt.figure()** to create a blank figure. Afterwards, we create a subplot in the blank figure, and then use the statement **px = data[input_country]** to put the data into graph figuring.

And the **px.plot(ax=ax, style='k-')** statement is to arrange the graph style, with 'k-' referring to a single black line, and this statement is the most important one in this program, figuring the chosen data.

Then we use **set_title**, **set_xlabel**, and **set_ylabel** to set the title and x,y labels form the graph, while the **ax.set_xticklabels(range(1980, 2016, 4), rotation=30, fontsize='small')** statement is to arrange the year tag to the x axis.

Afterwards, we set the name of the **save_figure_file**, and use **plt.show()** to present the figure directly.

The **judge(input_type)** function is to choose the right data (from data1 to data6) according to the type user inputs into the function, and giving out the figure result.

In the **comparison()** function, we are mainly repeating the process before.

Getting the user's input by **raw_input** function and then create a blank figure, put plots into the figure and then show and save the figure. What is worth mentioning is that we add several statements to set the y axis's upper and lower limits.

Firstly, we create a list for the maximum and minimum numbers of each country's data: **y_maxes = [], y_mins = []**.

Then in the for loop for the chosen countries, we get the max value of each chosen column of data, and append it into the blank list:

y_maxes.append(max(data[country])),

y_mins.append(min(data[country])).

Then we calculate the upper and lower limits with statements below:

```
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)
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)
```

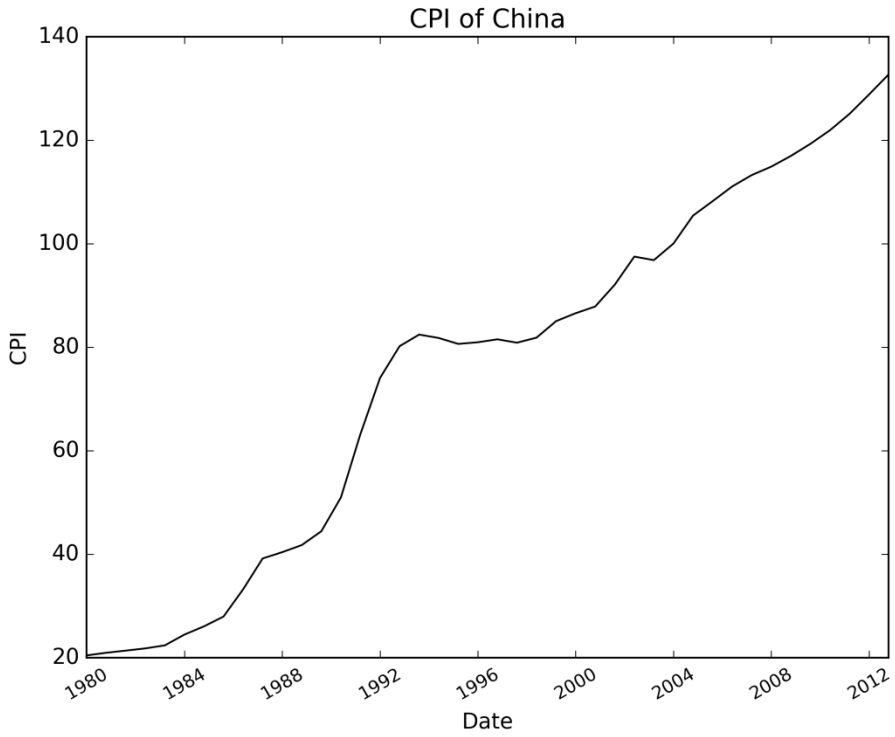
In this way, we get the upper and lower limits of y axis, and then we are able to draw the comparison figure.

(2) Results

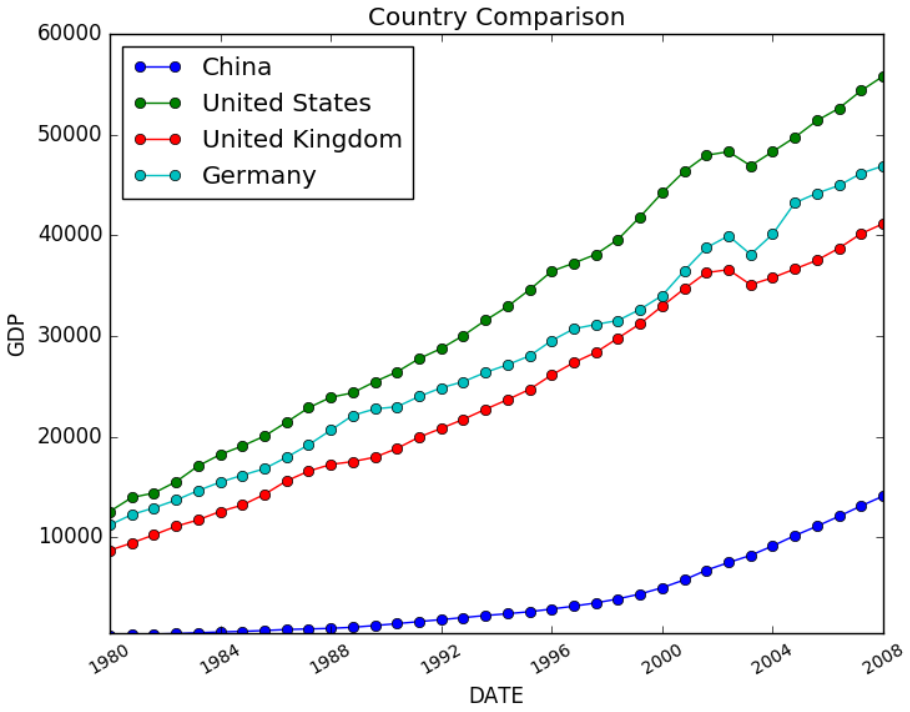
Put into different choices, this function will return different figures. In our trial, we put in "China", "CPI", and compare the GDP value among China, United States, United Kingdom and Germany.

The figures are as below:

Picture4. The CPI of China from 1980 to 2012



Picture5. The comparison of GDP between 4 countries from 1980 to 2008



8. Convert Images to Gif (dynamic effects)

(1) Scripts Interpretation

In this function, we focus on converting all the image files into one integrated gif file. And we add some restrictions to help resize and sort the image files.

Packages we use are os, PIL, and Images2gif.

Firstly, we create a blank list for images: **images = []**. Then we set the file path list: **width_and_file_paths = []**. Afterwards, we use a for loop to put all the paths and image files into the lists built before. These steps are for ranking and sorting, which will help us to convert and integrate the images according to the type user wants.

We design six types of sorting the images:

(a) Type 0: Resize the images to **max_width * max_height** and put them into the white-background figure, and then compound them after putting to the middle. All the figures here are in rectangular form.

(b) Type 1: Resize to **max_length**, put into white-background figure, and then compound them after putting to the middle. All the figures here are in quadratic form.

(c) Type 2: Put the images directly to the **white_background** figure sizing as **max_width * max_height**, and then compound them after putting to the middle. In rectangular form, but not resize the images.

(d) Type 3: Put the images directly to the **white_background** figure with **max_length** among all the images, and then compound them after putting to the middle. In rectangular form, but not resize the images.

(e) Type 4 : Compound the images directly ordered by width.

(f) Type 5 : Compound the images directly ordered by height.

Finally, we call this function and create the result file.

(2) Results

We create the GIF file and can dynamically display the time series data on the world map. As can be seen, when the time passed, we can see the changing of the color set when the country-level data has a significant alternation. Therefore, by inputting the images within a certain period, we can easily generate a dynamically visualized graph to show the changing of the data value.

9. Conclusions

This paper provides the data visualization method which can be achieved using Python, in terms of statistical plots, ranking process and map display. We show that Python can be compatible with a variety of packages and in different format of files to edit and save. We also display that by using the data visualization method, people who don’t have strong background knowledge of economics can also interpret the big data easily. From numerical data to visualized data is still a long process to achieve. We are still thinking a better way to achieve data visualization for implementing different strategies for a much simple way for user to experience and learn from data.

Our project still has the following ways for improvement:

(1) **Country Profile and the Ranking.py** method we first provide can be extended to build the GUI (Graphical User Interface). User can gain the knowledge simply by clicking a button or the country on the map and display the information immediately.

(2) For **GeoMap.py**, the threshold for each section of color class is a big problem. For how many classes of color set to categorize and for which color to implement matter a lot. If the range between the thresholds is inappropriate, the information displaying on the map will be unapparent. Therefore, we think that it's better to find more statistic and realistic ways to build the thresholds of each indicator.

(3) For **Country Trend and Comparison.py** method, we first extract six separate data files, and then judge the user input type and countries, this is not a convenient method because if user wants to search for other type of data, it will cost some time to find the data, but in this case, extracting the data separately before programming can save the running time for some extent. Another problem here is that every time the comparison image is shown in python, it is normal to exhibit, but when it is saved to the result folder, it only appears as blank, and then we have to save it manually. It is a problem we have not found proper solution.

(4) For **Convert to Gif.py** method, we mainly focus on deciding the sequence of the images, and the time of picture duration, as well as the frequency. After getting the gif file, the file has to be presented with Internet explorer to show its dynamic effects, or it will only be a static image. This is a restriction for gif file, we have not found a better way to show the gif file better.

10. References

[1] Wes, M. (2012). *Python for Data Analysis*. California: O'Reilly Media. ISBN: 9781449323592.

[2] Keven, S. (2014). *Introduction to Python for Econometrics, Statistics and Data Analysis*. [pdf] Available at: https://www.kevinsheppard.com/images/0/09/Python_introduction.pdf [Accessed 12 Sep. 2016].

[3] Wes, M. and PyData Development Team (2016). *pandas: powerful Python data analysis toolkit*. 1st ed. [pdf] Available at: http://pandas.pydata.org/pandas-docs/version/0.18.1/pandas.pdf [Accessed 1 Sep. 2016].

[4] IMF.org (2016). *The World Economic Outlook (WEO) Database, April 2016* [csv] Available at: https://www.imf.org/external/pubs/ft/weo/2016/01/weodata/index.aspx [Accessed 25 Aug. 2016].

[5] Wikipedia.org (2016). *List of national capitals in alphabetical order*, [online] Available at: https://en.wikipedia.org/ [Accessed 22 Aug. 2016].

[6] Unstats.un.org (2016) [online] Available at: http://unstats.un.org/unsd/methods/m49/m49regin.htm [Accessed 22 Aug. 2016].
[7] Nathan, Y. (2009). *How to Make a US County Thematic Map Using Free Tools*. Available at: http://flowingdata.com/ [Accessed 21 August, 2016].
[8] Amcharts.com (2006). Amcharts: Free SVG Maps [online] Available at: https://www.amcharts.com/svg-maps/ [Accessed 30 Aug. 2016].

11. Appendix

1. Country Profile and the Ranking.py

```
####Input and display our data file####
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)
#print f.head(190)[f.columns[1:2]]

####Build our dataset dictionary####
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
print sorted(data_dictionary().keys())

####Build a continent catigorized dictionary####
def countrymap():
    column = data_dictionary()
    country = column['Country by IMC'][0:190]
    continent=column["Continent"][0:190]
    print ("All the countries: ")
    for x in range(len(country)):
        print (str(x+1) + ". " + str(country[x]))
    sixcontinent=list(set(continent))
    #continent_map is a dictionary categorizing the coutries into six continents
    continent_map={}
    for k in sixcontinent:
        continent_map[k]=[]
    for s in range(len(continent)):
        for k in sixcontinent:
            if continent[s]==k:
                continent_map[k].append(country[s])
    print continent_map
    print "Total countries: %s" % len(country)
    return continent_map
```

```
####Build a ranking method####
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]

####Build a country profile class####
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####
if __name__ == "__main__":
    column = data_dictionary()
    country = column['Country by IMC']
    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
```

2. GeoMap.py

```
####Import the packages we need and set the global variables####
```

```
import csv
from bs4 import BeautifulSoup
import Image
from PIL import Image
from PIL import ImageDraw
import ImageFont
import svgwrite
import svgutils.transform as st
import string
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####
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

####Build 2 methods for 2 inputs: indicator and year####
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
input1=input1()
```

```
input2=input2()

####Edit the world map and save it to .txt format####
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.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-dash
array: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-dash
array: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-dash
array: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-dash
array: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"

worldmap()

####Convert .txt file to .jpg image####
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))
convert2jpg()

####Build the legend.jpg image####
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")

legend()

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

3. Country Trends and Comparison.py

```
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")

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

    # The function gives out the particular trend of one country.
    def single_trend(data):
        # Show the first 10 rows of the chosen data type
        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()

        # 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)

        judge(input_type)

#Call the country_trend function
country_trend()


# 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 = "result\GDP Comparison.jpg"
    elif chosen_type == "CPI":
        save_name = "result\CPI Comparison.jpg"
    elif chosen_type == "Net_debt":
        save_name = "result\Net Debt Comparison.jpg"
    elif chosen_type == "population":
        save_name = "result\Population Comparison.jpg"
    elif chosen_type == "employment":
        save_name = "result\Employment Comparison.jpg"
    elif chosen_type == "unemployment_rate":
        save_name = "result\Unemployment Rate Comparison.jpg"

    plt.show()
    plt.savefig(save_name)

comparison()
```

4. Convert Image to Gif.py

```
import os
from PIL import Image
import images2gif

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()

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

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)
```