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.

1. Building the environment
2. 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/>).

1. 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.

1. 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

1. 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

1. Case3

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

1. Build the dataset
2. 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.

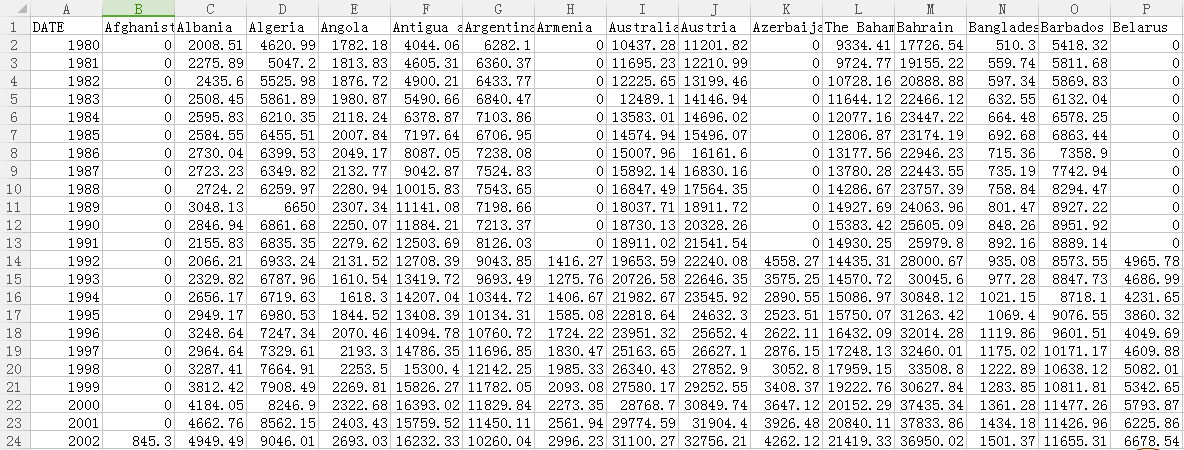
For countries’ basic information, the titles include **Country** (the name of the countries), **Country Code** (Alpha-2 code), **Alpha-3 code**, **Capital**, **Continent** (categorized into 6 continents: Africa, Asia, Europe, North America, South America and Oceania), **Area** (size of geographical area) 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).



1. 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.

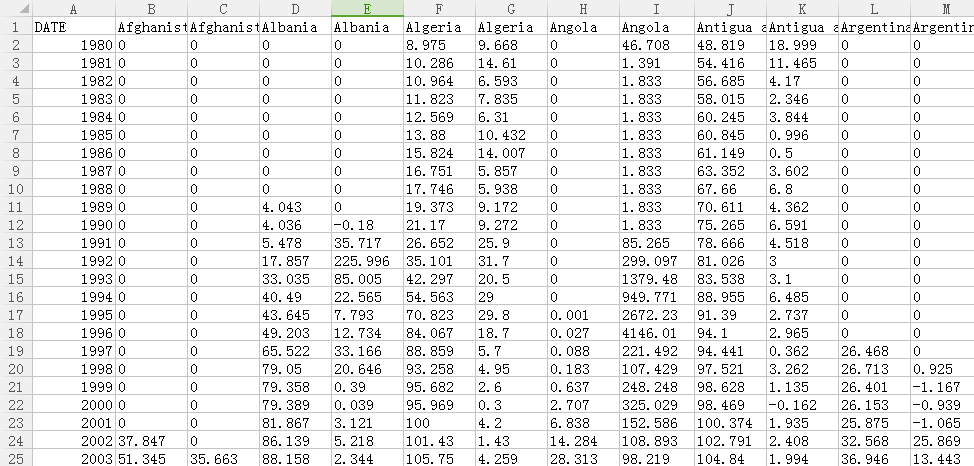
**Gross domestic product (GDP)** is a monetary [measure](https://en.wikipedia.org/wiki/Measurement_in_economics) of the market value of all final goods and services produced in a period, and the data we get here take one year as a period. Nominal GDP estimates we use here are the kind of data that is 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.



1. CPI.csv

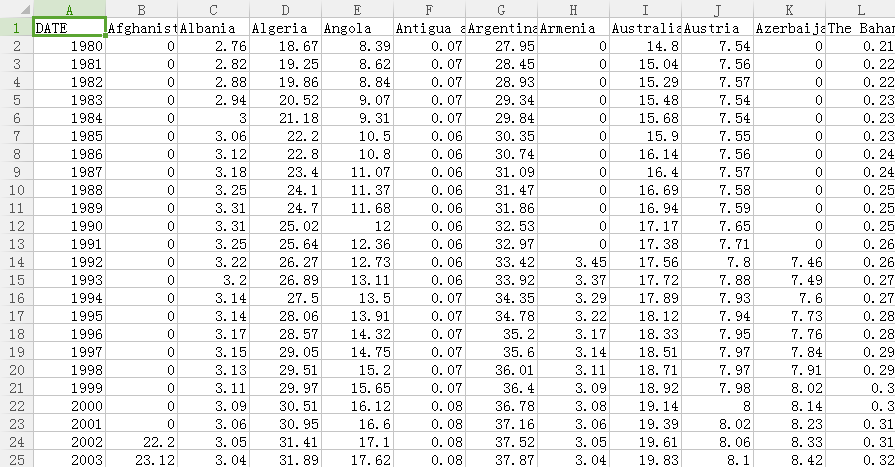
Same as “GDP.csv”, we get these CPI data of world GDP from IMF (International Monetary Fund) **The World Economic Outlook (WEO) April 2016** and **Wikipedia**. The original world data file contains a country’s basic information and also yearly data from 1980 to 2015, and some has predictions for 2016 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](https://en.wikipedia.org/wiki/Market_basket) of [consumer goods](https://en.wikipedia.org/wiki/Final_goods) and [services](https://en.wikipedia.org/wiki/Consumer_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.



1. Population.csv

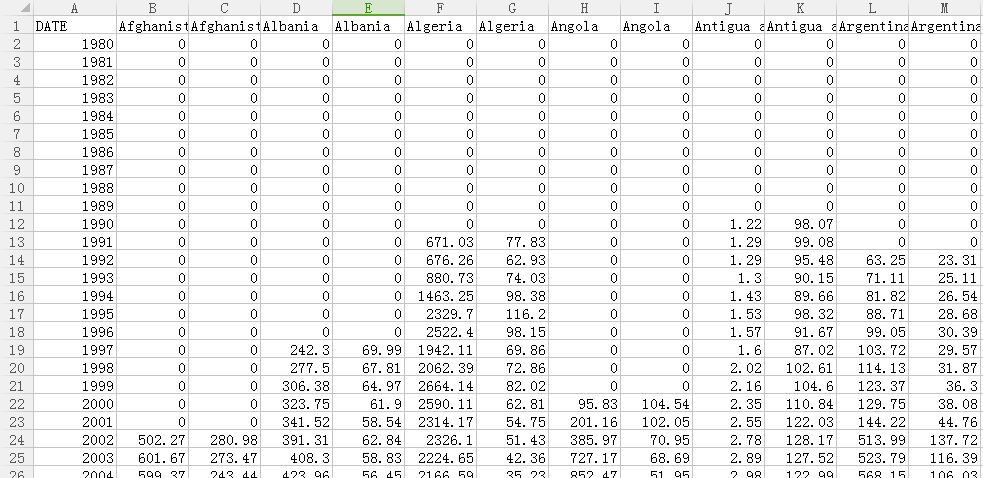
As the two files before, this Population.csv file is also extracted from the original file, and population refers to the total amount of people in one country.



1. 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.

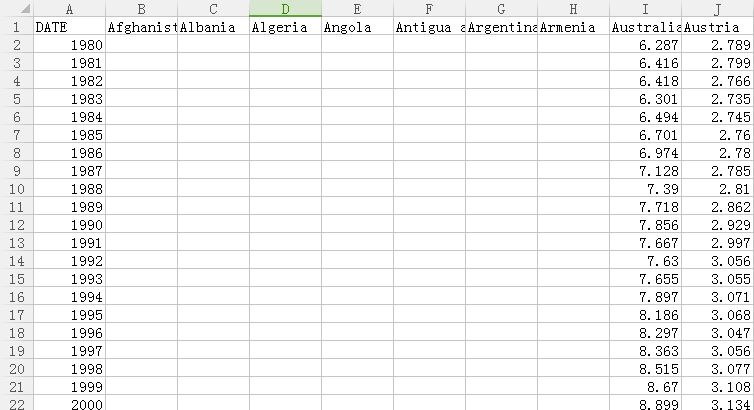
There are also “n/a” in the file, we also replace them with 0 to get a figure successfully.



1. 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.

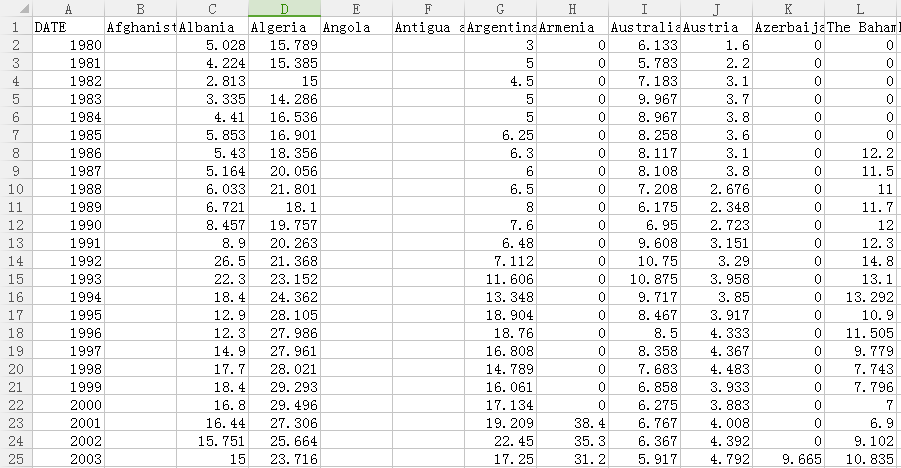
There are also default values in the file, we also replace them with 0. What must to be mentioned here is that there are several countries not having the employment rate data, just like China, and when this happens, we will get a blank figure containing nothing.



1. 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.



1. Worldmap.svg

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

1. Scripts and Results
2. **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) |

1. **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() |

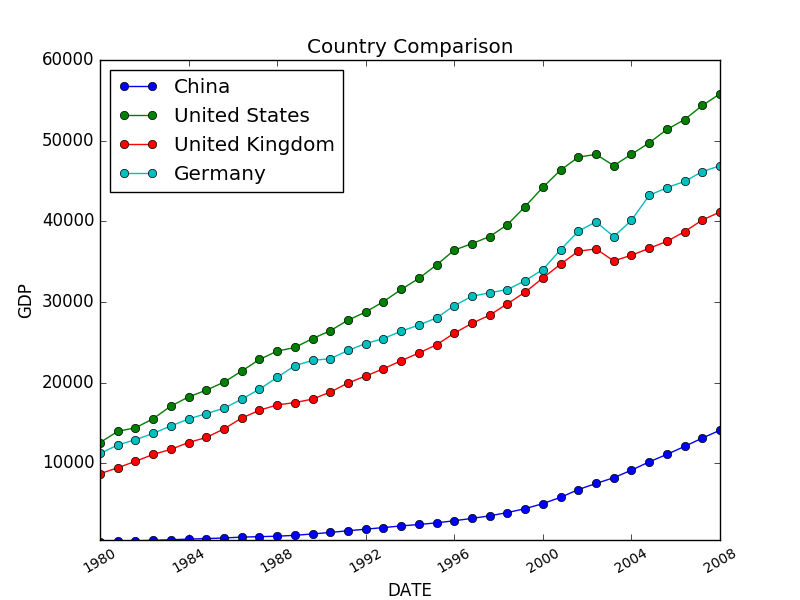
|  |  |
| --- | --- |
| **Convert .txt file to .jpg image** | |
| **Script** | **def** convert2jpg():  *#Revise the text file, in order to delete the <html>, <body>,</html>, and </body> in order for the QSvgRenderer to read only the text between<svg>to </svg>* f = open(**'%s\_%s.txt'** % (input1, input2), **'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()  D:\Freshman\申請文件\交換生\課程\Introductory to Python\Project\Economics data visualization with Python\result\world_Unemployment Rate_2006.jpg |

1. **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 + **" Unemploy****ment\_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()  China CPI |
| **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  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

****

1. **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. |

1. Q&A
2. 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.

1. 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.

1. 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.

1. 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.

1. 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](mailto:lisa5432126@gmail.com)) and X.Y, Lai ([laixiaoya96@163.com](mailto:laixiaoya96@163.com)). We will give you response as soon as possible.