Computer Science I CSCI-141
Analyzing Life Expectancy Data Project

11/13/2017

1 Problem Description

The World Bank (`https://www.worldbank.org`) provides free and open access to global development data.

For this project, you will analyze global data on life expectancy at birth, for the years 1960 through 2015. Additional information on what is meant by 'life expectancy at birth' can be found at `http://blogs.worldbank.org/opendata/what-does-life-expectancy-birth-really-mean`.

Your work for this project will include statistical analysis and graphing of the data.

Here is a summary of the problem tasks you will solve:

0. Utilities: This task involves writing tools for reading and processing the data, as well as defining data structures to store the data. The other tasks import and use these tools.
1. Ranking: This task involves processing the data and rank ordering it for a given year.
2. Growth: This task has inputs of a starting year and an ending year, and involves computing and rank ordering the absolute growth in life expectancy over the time period.
3. Drop: This task involves computing the largest drops in life expectancy experienced across any portion of the entire timeline.
4. Factors: This task involves looking at how region and income affect life expectancy. Turtle graphics is used to generate plots to visualize the data.

2 Getting Started

2.1 Data

The data is contained in two files:

- `worldbank_life_expectancy_data.txt`: this file contains the life expectancy data for countries and territories, as well as for some larger groupings.
- `worldbank_life_expectancy_metadata.txt`: this file contains metadata indicating which geographical region and income category each country or territory belongs to.

For the remainder of this document, the term *country* will be used to broadly refer to any country or territory listed in the provided data.

2.2 Provided Code

In addition to the two data files, two source files are provided for the project:

- `rit_lib.py`: required for data structures you will define.
- `test_overall.py`: provides testing for your programs.

The provided files **must not be changed**. You can download the data files and provided test module from:

`http://www.cs.rit.edu/~csci141/Projects/01/wbfiles.zip`.

The `rit_lib.py` file is not included in this download. You should have this file already, but it can also be downloaded from:

`http://www.cs.rit.edu/~csci141/lib/`.

2.3 Code to Write

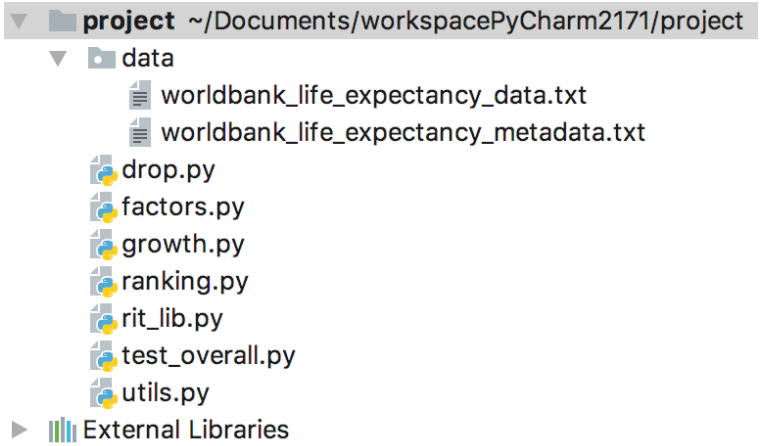
You will write the following programs *from scratch*:

0. `utils.py`: this supporting module includes data structure definitions and utility functions/tools used by the other programs; developing these utilities will be *Task 0*.
1. `ranking.py`: this is program *Task 1*.
2. `growth.py`: this is program *Task 2*.
3. `drop.py`: this is program *Task 3*.
4. `factors.py`: this is program *Task 4*.

2.4 Project Structure

You should create a new project using the provided zip file. You may choose to use *PyCharm* or some other means. Along with the source files, there should be a `data/` subdirectory with the data files.

If you use PyCharm, the layout of your project should eventually look similar to the following:



The location of your project may differ from the path shown in this example.

2.5 Standalone Execution versus Imported Execution

Each of your files must be able to run as a standalone program *and* as a module imported by another module, such as a test program.

When your file executes as a standalone program it should cause a `main()` function to be called and executed. When your file module is imported by another file, the `main()` function must not be called and executed. The way to achieve this is with the following program layout:

```
"""
file header identifies file name, description, author and date.
"""

# program imports, defined data structures, function definitions here

def main():
    """
    docstring for your main function here
    """
    # main's code body here...

if __name__ == '__main__':
    # main runs only when directly invoking this module
    main()

# end of program file
```

The conditional statement `if __name__ == '__main__'` will only evaluate to `True` when your program is run directly. When the code is imported by another module, this conditional statement will evaluate to `False` and the `main()` function will not be executed.

This organization allows you to include code that executes only when the program is run directly.

3 Data Files

In order to correctly read and process and filter the data, you will need to be aware of characteristics of the input data and metadata files.

3.1 worldbank_life_expectancy_data.txt

The main data file has the following format characteristics:

- The first line is header information that can be discarded.
- Each line contains fields that are comma-separated. There is a comma after the last field in the line as well. This last comma affects the result of splitting a string containing

a line of data using the string `split(',')` function. It causes an extra empty string to be included at the end of the generated list; you will want to discard that.

- A line of data includes a country name, followed by a capitalized three-letter country code, followed by floating point life expectancy values for the years from 1960 to 2015, inclusive. Note that some lines of data do not include life expectancy values for all years. Some lines of data do not include any life expectancy values at all. In these cases, there are successive commas with no characters between the commas.
- Note that some of the lines of data are not for specific countries, but rather for other groupings. For example, 'Least developed countries: UN classification'.

3.2 worldbank_life_expectancy_metadata.txt

The metadata file has the following format characteristics:

- The first line is header information that can be discarded.
- Each line contains fields that are comma-separated.
- A line of data includes a capitalized three-letter country code, followed by a region classification, followed by an income classification, possibly followed by special notes. If there are no notes, the line of data ends with a comma, and the notes are effectively an empty string following that. If there are notes, the notes are the last field. If the notes section itself contains a comma, the notes are placed within double-quotes.
- We are only interested in the first three pieces of data on each line: the country code, the region classification and the income classification. A line of input can be split using the comma character, and only the first three elements of the resulting list are of interest.
- Lines corresponding not to specific countries but rather to larger groupings have a country code, but do not have a region or income classification. In this case, there are commas with no characters between the commas.

4 The Tasks

For all tasks:

1. **The *modules* specified must be named exactly as described. Failure to do so will result in test failures and a low grade.** You must name your files exactly as specified in order for the test module to successfully import and use them.
2. **The user-defined *data structures* specified must be named and defined exactly as described. Failure to do so will result in test failures and a low grade.**
3. **The *functions* specified must be defined exactly as described. Failure to do so will result in test failures and a low grade.** You must follow the naming, parameter order, and return value specifications provided for the required functions. Failure to follow these specifications will cause the test module to fail.

You may of course add additional functions and data structures that you deem necessary to complete the tasks.

4.0 Task 0: Utility/Tools module `utils.py`

This file contains a set of utilities which includes data structures and functions used by the other program tasks.

4.0.1 Example Standalone Execution

When executed as a standalone program, your program should generate output similar to the following example. Note that the output below has been abbreviated for brevity.

```
Total number of entities: 263
Number of countries/territories: 217

Regions and their country count:
Middle East & North Africa: 21
Europe & Central Asia: 58
North America: 3
Latin America & Caribbean: 42
South Asia: 8
East Asia & Pacific: 37
Sub-Saharan Africa: 48

Income categories and their country count:
Lower middle income: 53
Upper middle income: 56
High income: 77
Low income: 31

Enter region name: South Asia
Countries in the 'South Asia' region:
India (IND)
Pakistan (PAK)
Nepal (NPL)
Bangladesh (BGD)
Bhutan (BTN)
Maldives (MDV)
Afghanistan (AFG)
Sri Lanka (LKA)

Enter income category: Upper middle income
Countries in the 'Upper middle income' income category:
Libya (LBY)
Turkey (TUR)
St. Lucia (LCA)
Malaysia (MYS)
```

```
Belarus (BLR)
<<<...abbreviated for brevity...>>>
Tuvalu (TUV)
Thailand (THA)

Enter name of country or country code (Enter to quit): Thailand
Data for Thailand:
Year: 1960  Life expectancy: 54.69931707
Year: 1961  Life expectancy: 55.23273171
Year: 1962  Life expectancy: 55.74521951
Year: 1963  Life expectancy: 56.23239024
Year: 1964  Life expectancy: 56.6942439
<<<...abbreviated for brevity...>>>
Year: 2014  Life expectancy: 74.42202439
Year: 2015  Life expectancy: 74.60119512

Enter name of country or country code (Enter to quit): NotACountry
'NotACountry' is not a valid country name or code

Enter name of country or country code (Enter to quit): BMU
Data for BMU:
Year: 1965  Life expectancy: 68.89780488
Year: 1970  Life expectancy: 70.29
Year: 1980  Life expectancy: 72.30463415
Year: 1991  Life expectancy: 74.0295122
Year: 2000  Life expectancy: 77.88536585
Year: 2001  Life expectancy: 77.88536585
Year: 2002  Life expectancy: 78.08780488
Year: 2003  Life expectancy: 78.33414634
Year: 2004  Life expectancy: 78.48536585
<<<...abbreviated for brevity...>>>
Year: 2014  Life expectancy: 80.79731707
Year: 2015  Life expectancy: 81.01219512

Enter name of country or country code (Enter to quit): ASM
Data for ASM:

Enter name of country or country code (Enter to quit):

Process finished with exit code 0
```

4.0.2 Required Standalone Behavior

When run standalone, your program must do the following:

- Read the data and metadata files. Do not prompt the user to enter the file name. You can hardcode the portion of the file name passed to the `read_data` function (see description below), as we will only be using the provided data and metadata files.
- Output the total number of entities in the data file (this number includes both individual countries and larger groupings).
- Output the total number of countries in the data file (this number excludes larger groupings).
- Output a summary of the different regions and the number of countries included in each region.
- Output a summary of the different income categories and the number of countries included in each category.
- Prompt the user to specify a region, and output the names and country codes of all countries in that region. If the user enters an invalid region, print an appropriate message and continue to the next requirement.
- Prompt the user to specify an income category, and output the names and country codes of all countries in that income category. If the user enters an invalid income category, print an appropriate message and continue to the next requirement.
- Enter a loop prompting the user for a country name or code. Inside this loop:
  - If the user enters a valid country name or country code, print out the life expectancy values for that country. Only print out existing values (any missing years are skipped).
  - If the user enters an invalid country name or country code, print an appropriate message.
  - If the user hits enter to quit, exit the loop.

4.0.3 Required Function Definitions

This module requires these functions:

1. **Name:** `read_data`

   Parameters: `filename`

   A *string*, giving the partial name of the data file. For this project, we are only using the two provided data files, and thus you will always pass the common filename base, `"worldbank_life_expectancy"` as the argument to this function. The function is responsible for prepending the path `"data/"` as well as appending the appropriate suffix to read both the main data and metadata file.

   Returned Result Type:

   One or more data structures representing the data contained in the main data and metadata files. You are encouraged to use Python dictionaries as well as user-defined data structures. You may return multiple data structures in the form of a tuple.

   Notes:

   You do not need to do any error checking for the provided file name. Your program may simply crash if the provided file name is not valid.

2. **Name:** `filter_region`

Parameters: `data, region`

The data parameter represents one or more data structures (possibly as a tuple) containing information from the data and metadata files. The region parameter is a *string* specifying a particular region by which to filter.

Returned Result Type:

One or more data structures representing data that has been filtered to only retain data corresponding to the specified region.

Notes:

If a valid region is entered, the original data should be filtered to retain only data corresponding to the specified region. If the special string `'all'` is entered, data for all regions should be retained, however, data for non-country larger groupings should still be discarded. If an invalid region is specified, all data is discarded. If the region specified is the empty string, this is considered to be an invalid region, and all data is discarded. **Note that regardless of the specified region, this function always filters out the non-countries (larger groupings).**

3. **Name:** `filter_income`

Parameters: `data, income`

The data parameter represents one or more data structures (possibly as a tuple) containing information from the data and metadata files. The income parameter is a *string* specifying a particular income category by which to filter.

Returned Result Type:

One or more data structures representing data that has been filtered to only retain data corresponding to the specified income category.

Notes:

If a valid income category is entered, the original data should be filtered to retain only data corresponding to the specified income category. If the special string `'all'` is entered, data for all income categories should be retained, however, data for non-country larger groupings should still be discarded. If an invalid income category is specified, all data is discarded. If the income category specified is the empty string, this is considered to be an invalid income category, and all data is discarded. **Note that regardless of the specified income category, this function always filters out the non-countries (larger groupings).**

Note that the details of how you store the information from the data files is being left up to you. You are encouraged to utilize Python dictionaries and your own data structures.

The return value from your `read_data` function must be acceptable input to either of your filtering functions. Your filtering functions must allowing chaining - that is, you must be able to perform one filtering operation, and use the output from that function as input to an additional filtering function.

4.1 Task 1: `ranking.py`

In this task you will write a program which ranks countries by their life expectancy for a particular year. The data may be filtered to consider only a particular region or income category, or combination thereof. Note that you will make use of your `utils.py` functions to complete this task.

4.1.1 Example Standalone Execution

When executed as a standalone program, your program should generate output similar to the following example.

```
Enter year of interest (-1 to quit): 1973
Enter region (type 'all' to consider all): Latin America & Caribbean
Enter income category (type 'all' to consider all): all

Top 10 Life Expectancy for 1973
1: Puerto Rico 72.47804878
2: Cuba 71.32095122
3: Aruba 70.13965854
4: Jamaica 69.15185366
5: Virgin Islands (U.S.) 68.9904878
6: Uruguay 68.88390244
7: Costa Rica 68.01746341
8: Argentina 67.46531707
9: Panama 67.05997561
10: Belize 66.91209756

Bottom 10 Life Expectancy for 1973
34: Bolivia 46.8785122
33: Haiti 48.32346341
32: Guatemala 54.07402439
31: Honduras 54.4307561
30: Nicaragua 55.47592683
29: Peru 55.758
28: El Salvador 55.82741463
27: Ecuador 59.15541463
26: Dominican Republic 60.14326829
25: Brazil 60.2515122

Enter year of interest (-1 to quit): 1988
Enter region (type 'all' to consider all): Middle East & North Africa
Enter income category (type 'all' to consider all): Lower middle income

Top 10 Life Expectancy for 1988
```

```
1: Syrian Arab Republic 69.47092683
2: Jordan 69.36680488
3: Tunisia 67.60790244
4: Egypt Arab Rep. 63.64056098
5: Morocco 63.48336585
6: Yemen Rep. 56.89634146
7: Djibouti 56.25592683

Bottom 10 Life Expectancy for 1988
7: Djibouti 56.25592683
6: Yemen Rep. 56.89634146
5: Morocco 63.48336585
4: Egypt Arab Rep. 63.64056098
3: Tunisia 67.60790244
2: Jordan 69.36680488
1: Syrian Arab Republic 69.47092683

Enter year of interest (-1 to quit): 2017
Valid years are 1960-2015

Enter year of interest (-1 to quit): 2013
Enter region (type 'all' to consider all): NotARegion
'NotARegion' is not a valid region

Enter year of interest (-1 to quit): -1

Process finished with exit code 0
```

4.1.2 Required Standalone Behavior

When run standalone, your program must do the following:

- Read the data and metadata files. Do not prompt the user to enter a file name. You can hardcode the portion of the file name passed to the `read_data` function, as we will only be using the provided data and metadata files.
- Enter a loop that prompts the user for a year, followed by a region, followed by an income category. If the year entered is -1, the program terminates. Otherwise, if any of the inputs is invalid or out of range, the loop restarts. An empty string is not considered a valid input for region or income category.
- Given valid inputs, the data is filtered to retain only the specified region and income category.
- Data for the specified year is extracted and sorted, and the top ten and bottom ten life expectancies for the given year are printed. If there are fewer than 20 total countries in the filtered data, the top ten and bottom ten lists may overlap somewhat or entirely.
- Additional requests are processed until the user enters -1 for the year.

4.1.3 Required Data Structure and Function Definitions

This module includes one required data structure and one required function. Note that the required data structure should be contained in the `utils.py` file.

1. `CountryValue` data structure with these components:
   - `country`: a *string* representing a country.
   - `value`: a *float* representing a life expectancy value for that country.

2. **Name:** `sorted_ranking_data`
   Parameters: `data, year`
   The data parameter represents one or more data structures (possibly as a tuple) containing information from the data and metadata files. The year parameter is an integer representing the year under consideration.
   Returned Result Type:
   A list of `CountryValue` structures, sorted in descending order (highest to lowest).
   Notes:
   Only countries that contain data for the specified year are included in the sorted list. For this, and any future tasks involving sorting, you are free to use any sorting functionality (built-in or otherwise).

4.2 Task 2: `growth.py`

In this task you will write a program which ranks countries by absolute growth in life expectancy over a specified range of years. The data may be filtered to consider only a particular region or income category, or combination thereof. Note that you will make use of your `utils.py` functions to complete this task.

4.2.1 Example Standalone Execution

When executed as a standalone program, your program should generate output similar to the following example.

```
Enter starting year of interest (-1 to quit): 1970
Enter ending year of interest (-1 to quit): 1976
Enter region (type 'all' to consider all): Sub-Saharan Africa
Enter income category (type 'all' to consider all): all

Top 10 Life Expectancy Growth: 1970 to 1976
1: Senegal 5.41978048
2: Gambia 5.182707309999998
3: Gabon 4.959341469999998
4: Cote d'Ivoire 4.68143903
5: Central African Republic 4.67256098
6: Sierra Leone 4.458926829999996
```

```
7: Mali 4.240073170000002
8: Liberia 4.165780490000003
9: Cabo Verde 3.93004878
10: Botswana 3.7557804899999994

Bottom 10 Life Expectancy Growth: 1970 to 1976
47: Uganda 0.5273414600000024
46: Rwanda 0.9763170700000003
45: Ethiopia 1.1726829300000006
44: Niger 1.3347073199999997
43: Congo Dem. Rep. 1.502439019999997
42: Sudan 1.57292683
41: Congo Rep. 1.7168780400000045
40: Mauritius 1.737585369999998
39: Equatorial Guinea 1.7567317099999968
38: Ghana 1.780487799999996

Enter starting year of interest (-1 to quit): -1

Process finished with exit code 0
```

4.2.2 Required Standalone Behavior

When run standalone, your program must do the following:

- Read the data and metadata files. Do not prompt the user to enter a file name. You can hardcode the portion of the file name passed to the `read_data` function, as we will only be using the provided data and metadata files.
- Enter a loop that prompts the user for a starting year, followed by an ending year, followed by a region, followed by an income category. If either the starting or ending year entered is -1, the program terminates. Otherwise, if any of the inputs is invalid or out of range, the loop restarts. An empty string is not considered a valid input for region or income category.
- Given valid inputs, the data is filtered to retain only the specified region and income category.
- Data for the specified years is extracted and the growth values sorted, and the top ten and bottom ten life expectancy growth values over the given range of years are printed. If there are fewer than 20 total countries in the filtered data, the top ten and bottom ten lists may overlap somewhat or entirely.
- Additional requests are processed until the user enters -1 for the starting or ending year.

4.2.3 Required Data Structure and Function Definitions

This module includes one required data structure and one required function. The required data structure is the `CountryValue` data structure described previously for Task 1.

1. **Name:** `sorted_growth_data`
   Parameters: `data, year1, year2`
   The data parameter represents one or more data structures (possibly as a tuple) containing information from the data and metadata files. The year1 parameter is an integer representing the starting year under consideration. The year2 parameter is an integer representing the ending year under consideration.
   Returned Result Type:
   A list of `CountryValue` structures, sorted in descending order (highest to lowest). The value stored in this structure is the absolute growth in life expectancy for the country between the starting year and the ending year.
   Notes:
   Only countries that contain data for both the specified starting and ending year are included in the sorted list.

4.3 Task 3: `drop.py`

In this task you will write a program which identifies the 10 worst drops in life expectancy throughout the 1960-2015 timeframe. The data is only filtered to remove non-country (larger groupings) entries. Note that you will make use of your `utils.py` functions to complete this task.

4.3.1 Example Standalone Execution

When executed as a standalone program, your program should generate output similar to the following example.

```
Worst life expectancy drops: 1960 to 2015
1: Rwanda from 1984 (50.78063415) to 1993 (27.07890244): -23.701731709999997
2: Cambodia from 1968 (42.53968293) to 1977 (19.2655122): -23.274170729999998
3: Zimbabwe from 1986 (61.92531707) to 2002 (40.67914634): -21.246170729999996
4: Lesotho from 1991 (59.63639024) to 2004 (43.53346341): -16.102926829999994
5: Botswana from 1988 (62.94563415) to 2001 (48.61031707): -14.335317079999996
6: Swaziland from 1990 (59.34519512) to 2004 (45.74534146): -13.59985366
7: South Africa from 1992 (62.32517073) to 2005 (51.55734146): -10.76782927
8: Zambia from 1978 (51.64119512) to 1996 (42.02480488): -9.616390240000001
9: Kenya from 1986 (59.64873171) to 2000 (50.78517073): -8.863560980000003
10: Namibia from 1991 (61.26912195) to 2002 (54.10509756): -7.164024390000002

Process finished with exit code 0
```

4.3.2 Required Standalone Behavior

When run standalone, your program must do the following:

- Read the data and metadata files. Do not prompt the user to enter a file name. You can hardcode the portion of the file name passed to the `read_data` function, as we will only be using the provided data and metadata files.
- Filter the data to retain all regions and income categories (but to remove non-country entries).
- Data from the entire 1960 to 2015 timeframe is analyzed. The largest drop in life expectancy is computed for each country. These values are sorted and the largest 10 drops are printed.

4.3.3 Required Data Structure and Function Definitions

This module includes one required data structure and one required function. Note that the required data structure should be contained in the `utils.py` file.

1. `Range` data structure with these components:
   - `country`: a *string* representing a country.
   - `year1`: an *integer* representing a first year.
   - `year2`: an *integer* representing a second year.
   - `value1`: a *float* representing a first life expectancy value for that country corresponding to the first year.
   - `value2`: a *float* representing a second life expectancy value for that country corresponding to the second year.

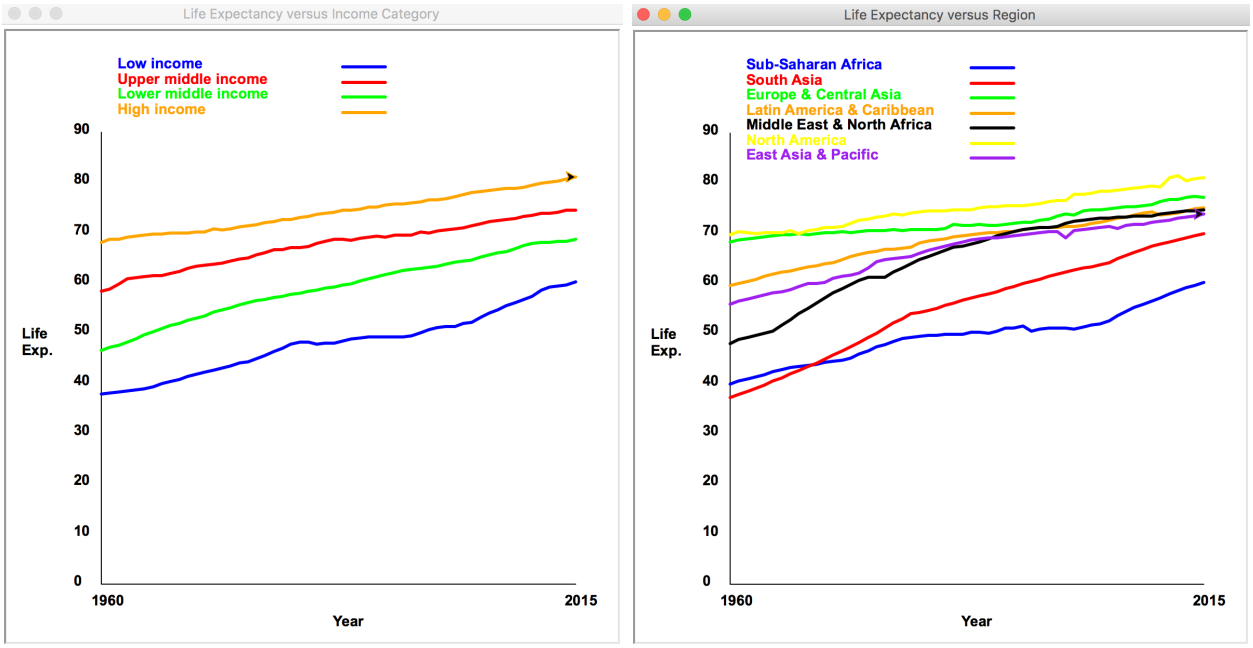
2. **Name:** `sorted_drop_data`
   Parameters: `data`
   The data parameter represents one or more data structures (possibly as a tuple) containing information from the data and metadata files.
   Returned Result Type:
   A sorted list of `Range` structures. The list is sorted in ascending order based on the change in life expectancy from the first value to the second value. The first year should be the beginning year of the interval that generated the largest drop in life expectancy. The second year should be the ending year of the interval that generated the largest drop in life expectancy. Note that a drop in life expectancy results in a negative value when computing `value2 - value1`. Thus the list is sorted in ascending order and the largest drop appears first in the list.
   Notes:
   Only countries that contain data for at least two years are considered. This function identifies, for each country, the starting and ending year that yield the maximum drop in life expectancy for that country between the starting and ending year. The starting year must precede the ending year.

4.4 Task 4: `factors.py`

In this task you will write a program which will compute one measure of the effect that income category and geographical region have on life expectancy. You will use turtle graphics to produce a visualization of this data.

4.4.1 Example Standalone Execution

When executed as a standalone program, your program should generate graphs similar to the following:



4.4.2 Required Standalone Behavior

When run standalone, your program must do the following:

- Read the data and metadata files. Do not prompt the user to enter a file name. You can hardcode the portion of the file name passed to the `read_data` function, as we will only be using the provided data and metadata files.
- Filter the data to retain all income categories (but remove non-country entries).
- For each income category, compute the median life expectancy for each year.
- Use turtle graphics to generate a line plot to show how the median life expectancy has changed from 1960 to 2015 for countries belonging to each income category.
- Prompt the user to hit enter to continue.
- Repeat this analysis and generate a plot for the different region classifications.
- Leave the window open until the user clicks to close the window.

4.4.3 Required Data Structure and Function Definitions

There are no required data structures or functions for this task. You will want one or more functions that take a filtered data set and compute the median life expectancy among the countries present for each year.

When there are an odd number of data points, the median is the value found at the middle index if the data are sorted. For an even number of data points, you may compute the median in any reasonable way, including using the 'lower' median, the 'upper' median, or the average of the 'upper' and 'lower' median.

As an example, if you are considering the South Asia region, there are 8 countries in that region. For each year, you will have as many as 8 life expectancy values (there may be fewer if one or more of the countries did not report a value for that year). You will compute the median of those values, and that median value will be used to represent the life expectancy for the South Asia region for that year.

Clearly, this is not a perfect measure, as it treats equally all countries, regardless of the number of people that live in that country.

We have not specified what should happen graphically if there were a year for which there were no data for an entire region or entire income category. For the required plots, this situation does not occur. You should implement *some* strategy should this situation arise, but the specific strategy is not important.

For this task, it is reasonable to use specific hard-coded values for plotting distances. **You should, however, make your plotting function general enough that a single plotting function suffices for both plotting the income-based graph as well as the region-based graph.**

You may use functions like `setpos`, and you may specify a list of colors to choose from, knowing that you need at most seven for either of these particular graphs.

You should aim for about the level of detail provided in the examples. Your axes do need labels and some values marked, but tick marks are not required. You do need to generate a legend so that the lines in the graph can be identified.

5 Testing Your Programs

The test file, `test_overall.py`, has been provided to you to test your implementation of various portions of this project. When you run the `test_overall.py` program, it will import your other modules and perform verbose pass-fail tests on components of those programs.

Please make sure that your program modules *implement the interface* required by the test program.

Do not change the interface of any of the required functions because similar test programs will be used to verify your code.

Your program will be tested against other, different, previously unseen, test cases, which are based on the same data files as those provided.

6 Submission Requirements

Please submit your code in a file called `abc1234.zip`, where `abc1234` is your **RIT username**. This file must contain *only* the five files written to complete Tasks 0-4. The zip file structure must be "flat"; that means there must be **no folders within the zip file**. Click on the individual files to simultaneously select them all, and zip them up. Do not use any compression mechanism other than zip. Submit your `abc1234.zip` file to the MyCourses Project dropbox.

See the **Grading** section for penalties for failure to follow the submission requirements.

7 Grading

7.1 Grading Distribution

Here is the grading distribution for the implementation. Make sure that you pay attention to what is required for standalone execution, in addition to testing with the provided test program.

- 40% Task 0: Functionality of components, including standalone execution, in `utils.py`;
- 10% Task 1: Functionality of components, including standalone execution, in `ranking.py`;
- 10% Task 2: Functionality of components, including standalone execution, in `growth.py`;
- 10% Task 3: Functionality of components, including standalone execution, in `drop.py`;
- 20% Task 4: Functionality of components in `factors.py`; and
- 10% Documentation and Style:
  Each file must have a *header docstring* containing the file name and your name. (Remember, docstrings have 'triple quote' syntax).
  Each function must have its own, *function docstring* with the following content:
  - description of input parameters and types;
  - description of the function's return value(s) and types; and
  - description of pre-conditions or post-conditions if appropriate.

  Further, the code should be well organized on the page. There should be at least one blank line between each component (i.e., between data structure definitions and between functions).

7.2 Additional Grading Penalties

It is possible that you got everything working but failed to follow directions, making testing and grading very difficult, error-prone and time-consuming. Also it is possible that you got everything working but cheated to succeed.

This leads to these *possible deductions over and above the distribution described earlier.*

1. 100% If **cheat-checking** detects that your modules are strongly similar to those of others, you will be subject to RIT's Academic Honesty Policy as detailed in `http://www.cs.rit.edu/~csci141/syllabus.html`. That means you will receive a 0 grade, and all parties to the cheating incident will receive this treatment. If the infraction is more serious, there may be further penalties; see the syllabus for details.

2. 100% Your programs must run using Python version 3. If you wrote using Python version 2 you will fail this assignment. **Python version 3 is the only acceptable interpreter**. Make sure your programming environment is using the correct dialect.
3. 50% If your modules do not work with the provided test program, you will fail all the tests.

   You must use the provided testing code to test your code and ensure that you follow the interface with your implementations. If the test program cannot execute your functions, or if your functions cause the test program to crash or malfunction, *then you will fail this assignment*.

   You must name the specified functions *exactly as used in the provided test program*. If you fail to name the functions exactly as stated, you have broken the interface specification, and *you will fail this assignment*.
4. 5% If you fail to follow the *Submission Steps to Follow for a Flat Zip* instructions exactly, you will receive a 5% penalty off the top of your earned grade. You must not include anything in your zip file that was not specified in the submission section.
5. 20% This project has the standard 8-hour late dropbox policy used for lab submissions. If you submit to the dropbox after the deadline as a "late submission", you will receive a 20% penalty. This is calculated as 20% deduction from your *earned score before the penalty*.