**Aggregating on index levels/fields**

If you have a DataFrame with a multi-level row index, the individual levels can be used to perform the groupby. This allows advanced aggregation techniques to be applied along one or more levels in the index and across one or more columns.

In this exercise you'll use the full Gapminder dataset which contains yearly values of life expectancy, population, child mortality (per 1,000) and per capita gross domestic product (GDP) for every country in the world from 1964 to 2013.

Your job is to create a multi-level DataFrame of the columns 'Year', 'Region' and 'Country'. Next you'll group the DataFrame by the 'Year' and 'Region' levels. Finally, you'll apply a dictionary aggregation to compute the total population, spread of per capita GDP values and average child mortality rate.

The Gapminder CSV file is is available as 'gapminder.csv'.

**INSTRUCTIONS**

* Read 'gapminder.csv' into a DataFrame with index\_col=['Year','region','Country']. Sort the index.
* Group gapminder with a level of ['Year','region'] using its level parameter. Save the result as by\_year\_region.
* Define the function spread which returns the maximum and minimum of an input series.
* Create a dictionary with 'population':'sum', 'child\_mortality':'mean' and 'gdp':spread as aggregator. This has been done for you.
* Use the aggregator dictionary to aggregate by\_year\_region. Save the result as aggregated.
* Print the last 6 entries of aggregated.

# Read the CSV file into a DataFrame and sort the index: gapminder

gapminder = pd.read\_csv('gapminder.csv', index\_col=['Year', 'region', 'Country']).sort\_index()

# Group gapminder by 'Year' and 'region': by\_year\_region

by\_year\_region = gapminder.groupby(level=['Year', 'region'])

# Define the function to compute spread: spread

def spread(series):

return series.max() - series.min()

# Create the dictionary: aggregator

aggregator = {'population':'sum', 'child\_mortality':'mean', 'gdp':spread}

# Aggregate by\_year\_region using the dictionary: aggregated

aggregated = by\_year\_region.agg(aggregator)

# Print the last 6 entries of aggregated

print(aggregated.tail(6))