# Grouping by multiple columns

In this exercise, you will return to working with the Titanic dataset and use .groupby() to analyze the distribution of passengers who boarded the Titanic.

The 'pclass' column identifies which class of ticket was purchased by the passenger and the 'embarked' column indicates at which of the three ports the passenger boarded the Titanic. 'S' stands for Southampton, England, 'C' for Cherbourg, France and 'Q' for Queenstown, Ireland.

Your job is to first group by the 'pclass' column and count the number of rows in each class using the 'survived' column. You will then group by the 'embarked' and 'pclass' columns and count the number of passengers.

Create a DataFrame titanic from titanic.csv file

##### INSTRUCTIONS

* Group by the 'pclass' column and save the result as by\_class.
* Aggregate the 'survived' column of by\_class using .count(). Save the result as count\_by\_class.
* Print count\_by\_class. This has been done for you.
* Group titanic by the 'embarked' and 'pclass' columns. Save the result as by\_mult.
* Aggregate the 'survived' column of by\_mult using .count(). Save the result as count\_mult.
* Print count\_mult.

# Grouping by another series

In this exercise, you'll use two data sets from [**Gapminder.org**](http://gapminder.org/) to investigate the average life expectancy (in years) at birth in 2010 for the 6 continental regions. To do this you'll read the life expectancy data per country into one pandas DataFrame and the association between country and region into another.

By setting the index of both DataFrames to the country name, you'll then use the region information to group the countries in the life expectancy DataFrame and compute the mean value for 2010.

The life expectancy CSV file is available to you life\_expectancy.csv and the regions filename is available in regios.csv

##### INSTRUCTIONS

* Read life\_fname into a DataFrame called life and set the index to 'Country'.
* Read regions\_fname into a DataFrame called regions and set the index to 'Country'.
* Group life by the region column of regions and store the result in life\_by\_region.
* Print the mean over the 2010 column of life\_by\_region.

# Computing multiple aggregates of multiple columns

The .agg() method can be used with a tuple or list of aggregations as input. When applying multiple aggregations on multiple columns, the aggregated DataFrame has a multi-level column index.

In this exercise, you're going to group passengers on the Titanic by 'pclass' and aggregate the 'age' and 'fare' columns by the functions 'max' and 'median'. You'll then use multi-level selection to find the oldest passenger per class and the median fare price per class.

Use the titanic DataFrame.

##### INSTRUCTIONS

* Group titanic by 'pclass' and save the result as by\_class.
* Select the 'age' and 'fare' columns from by\_class and save the result as by\_class\_sub.
* Aggregate by\_class\_sub using 'max' and 'median'. You'll have to pass 'max' and 'median' in the form of a list to .agg().
* Use .loc[] to print all of the rows and the column specification ('age','max').
* Use .loc[] to print all of the rows and the column specification ('fare','median').

# 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.

# Grouping on a function of the index

Groubpy operations can also be performed on transformations of the index values. In the case of a DateTimeIndex, we can extract portions of the datetime over which to group.

In this exercise you'll read in a set of sample sales data from February 2015 and assign the 'Date' column as the index. Your job is to group the sales data by the day of the week and aggregate the sum of the 'Units' column.

Is there a day of the week that is more popular for customers? To find out, you're going to use .strftime('%a') to transform the index datetime values to abbreviated days of the week.

The sales data CSV file is available to you as 'sales.csv'.

##### INSTRUCTIONS

* Read 'sales.csv' into a DataFrame with index\_col='Date'and parse\_dates=True.
* Create a groupby object with sales.index.strftime('%a') as input and assign it to by\_day.
* Aggregate the 'Units' column of by\_day with the .sum()method. Save the result as units\_sum.
* Print units\_sum.

# Detecting outliers with Z-Scores

using the zscorefunction, you can apply a .transform() method after grouping to apply a function to groups of data independently. The z-score is also useful to find outliers: a z-score value of +/- 3 is generally considered to be an outlier.

In this example, you're going to normalize the Gapminder data in 2010 for life expectancy and fertility by the z-score per region. Using boolean indexing, you will filter out countries that have high fertility rates and low life expectancy for their region.

The Gapminder DataFrame for 2010 indexed by 'Country' is provided for you as gapminder1\_2010.csv

##### INSTRUCTIONS

* mport zscore from scipy.stats.
* Group gapminder\_2010 by 'region' and transform the ['life','fertility'] columns by zscore.
* Construct a boolean Series of the bitwise or between standardized['life'] < -3 and standardized['fertility'] > 3.
* Filter gapminder\_2010 using .loc[] and the outliersBoolean Series. Save the result as gm\_outliers.
* Print gm\_outliers

# Filling missing data (imputation) by group

Many statistical and machine learning packages cannot determine the best action to take when missing data entries are encountered. Dealing with missing data is natural in pandas (both in using the default behavior and in defining a custom behavior). In Chapter 1, you practiced using the .dropna() method to drop missing values. Now, you will practice imputing missing values. You can use .groupby() and .transform() to fill missing data appropriately for each group.

Your job is to fill in missing 'age' values for passengers on the Titanic with the median age from their 'gender' and 'pclass'. To do this, you'll group by the 'sex' and 'pclass' columns and transform each group with a custom function to call .fillna() and impute the median value.

Use the DataFrame titanic. Explore it in the IPython Shell by printing the output of titanic.tail(10). Notice in particular the NaNs in the 'age' column.

##### INSTRUCTIONS

* Group titanic by 'sex' and 'pclass'. Save the result as by\_sex\_class.
* Write a function called impute\_median() that fills missing values with the median of a series. This has been done for you.
* Call .transform() with impute\_median on the 'age'column of by\_sex\_class.
* Print the output of titanic.tail(10).

# Other transformations with .apply

The .apply() method when used on a groupby object performs an arbitrary function on each of the groups. These functions can be aggregations, transformations or more complex workflows. The .apply() method will then combine the results in an intelligent way.

In this exercise, you're going to analyze economic disparity within regions of the world using the Gapminder data set for 2010. To do this you'll define a function to compute the aggregate spread of per capita GDP in each region and the individual country's z-score of the regional per capita GDP. You'll then select three countries - United States, Great Britain and China - to see a summary of the regional GDP and that country's z-score against the regional mean.

Create DataFrame as gapminder\_2010 from gapminder1\_2010.csv

The following function has been defined for your use:

def disparity(gr):

# Compute the spread of gr['gdp']: s

s = gr['gdp'].max() - gr['gdp'].min()

# Compute the z-score of gr['gdp'] as (gr['gdp']-gr['gdp'].mean())/gr['gdp'].std(): z

z = (gr['gdp'] - gr['gdp'].mean())/gr['gdp'].std()

# Return a DataFrame with the inputs {'z(gdp)':z, 'regional spread(gdp)':s}

return pd.DataFrame({'z(gdp)':z , 'regional spread(gdp)':s})

##### INSTRUCTIONS

* Group gapminder\_2010 by 'region'. Save the result as regional.
* Apply the provided disparity function on regional, and save the result as reg\_disp.
* Use .loc[] to select ['United States','United Kingdom','China'] from reg\_disp and print the results.

# Grouping and filtering with .apply()

By using .apply(), you can write functions that filter rows within groups. The .apply() method will handle the iteration over individual groups and then re-combine them back into a Series or DataFrame.

In this exercise you'll take the Titanic data set and analyze survival rates from the 'C' deck, which contained the most passengers. To do this you'll group the dataset by 'sex' and then use the .apply() method on a provided user defined function which calculates the mean survival rates on the 'C'deck:

def c\_deck\_survival(gr):

c\_passengers = gr['cabin'].str.startswith('C').fillna(False)

return gr.loc[c\_passengers, 'survived'].mean()

Use the existing DataFrame titanic.

##### INSTRUCTIONS

* Group titanic by 'sex'. Save the result as by\_sex.
* Apply the provided c\_deck\_survival function on the by\_sex DataFrame. Save the result as c\_surv\_by\_sex.
* Print c\_surv\_by\_sex.

# Grouping and filtering with .filter()

You can use groupby with the .filter() method to remove whole groups of rows from a DataFrame based on a boolean condition.

In this exercise, you'll take the February sales data and remove entries from companies that purchased less than 35 Units in the whole month.

First, you'll identify how many units each company bought for verification. Next you'll use the .filter() method after grouping by 'Company' to remove all rows belonging to companies whose sum over the 'Units' column was less than 35. Finally, verify that the three companies whose total Units purchased were less than 35 have been filtered out from the DataFrame.

##### INSTRUCTIONS

* Group sales by 'Company'. Save the result as by\_company.
* Compute and print the sum of the 'Units' column of by\_company.
* Call .filter() on by\_company with lambda g:g['Units'].sum() > 35 as input and print the result.

# Filtering and grouping with .map()

You have seen how to group by a column, or by multiple columns. Sometimes, you may instead want to group by a function/transformation of a column. The key here is that the Series is indexed the same way as the DataFrame. You can also mix and match column grouping with Series grouping.

In this exercise your job is to investigate survival rates of passengers on the Titanic by 'age' and 'pclass'. In particular, the goal is to find out what fraction of children under 10 survived in each 'pclass'. You'll do this by first creating a boolean array where True is passengers under 10 years old and False is passengers over 10. You'll use .map() to change these values to strings.

Finally, you'll group by the under 10 series and the 'pclass'column and aggregate the 'survived' column. The 'survived' column has the value 1 if the passenger survived and 0 otherwise. The mean of the 'survived' column is the fraction of passengers who lived.

Use the DataFrame titanic.

##### INSTRUCTIONS

* Create a Boolean Series of titanic['age'] < 10 and call .map with {True:'under 10', False:'over 10'}.
* Group titanic by the under10 Series and then compute and print the mean of the 'survived' column.
* Group titanic by the under10 Series as well as the 'pclass' column and then compute and print the mean of the 'survived' column.