Loading Olympic edition DataFrame

1) In this chapter, you'll be using **The Guardian's Olympic medal dataset**.
   a) Your first task here is to prepare a DataFrame `editions` from a *tab-separated values* (TSV) file.
   b) Initially, `editions` has 26 rows (one for each Olympic *edition*, i.e., a year in which the Olympics was held) and 7 columns: `'Edition'`, `'Bronze'`, `'Gold'`, `'Silver'`, `'Grand Total'`, `'City'`, and `'Country'`.For the analysis that follows, you won't need the overall medal counts, so you want to keep only the useful columns from `editions`: `'Edition'`, `'Grand Total'`, `City`, and `Country`.

Input Sample:

	Edition	Bronze	Gold	Silver	Grand Total	City	Country
0	1896	40	64	47	151	Athens	Greece
1	1900	142	178	192	512	Paris	France
2	1904	123	188	159	470	St. Louis	UnitedStates
3	1908	211	311	282	804	London	UnitedKingdom
4	1912	284	301	300	885	Stockholm	Sweden

INSTRUCTIONS

- Read `file_path` into a DataFrame called `editions`. The identifier `file_path` has been pre-defined with the filename `'SOM_EDITIONS.tsv'`. You'll have to use the option `sep='\t'` because the file uses tabs to delimit fields (`pd.read_csv()` expects commas by default).
- Select only the columns `'Edition'`, `'Grand Total'`, `'City'`, and `'Country'` from `editions`.
- Save the modified dataset (i.e editions) in following directory: **"/code/ex1a_eval.csv"**

Sample Output:

	Edition	Grand Total	City	Country
0	1896	151	Athens	Greece
1	1900	512	Paris	France
2	1904	470	St. Louis	United States
3	1908	804	London	United Kingdom
4	1912	885	Stockholm	Sweden

```
5  | 1920  | 1298      | Antwerp    | Belgium
```

Loading IOC codes DataFrame

2) Your task here is to prepare a DataFrame `ioc_codes` from a comma-separated values (CSV) file.
    a) Initially, ioc_codes has 200 rows (one for each country) and columns: 'Country', 'NOC', & 'ISO code'.
    b) For the analysis that follows, you want to keep only the useful columns from `ioc_codes`: `'Country'` and `'NOC'` (the column `'NOC'`contains three-letter codes representing each country).

Input sample:

```
        Country          NOC    ISO code
0   | Afghanistan       | AFG   | AF
1   | Albania           | ALB   | AL
2   | Algeria           | ALG   | DZ
3   | American Samoa*   | ASA   | AS
4   | Andorra           | AND   | AD
```

INSTRUCTIONS

- Read `file_path` into a DataFrame called `ioc_codes`. The identifier `file_path` has been pre-defined with the filename `IOC_COUNTRY_CODES.csv'`.
- Select only the columns `'Country'` and `'NOC'` from `ioc_codes`.
- Print the modified `ioc_codes` DataFrame

Building medals DataFrame

3) Here, you'll start with the DataFrame `editions` from the previous exercise.

    a) You have a sequence of files `summer_1896.csv`, `summer_1900.csv`, ..., `summer_2008.csv`, one for each Olympic edition (year).You will build up a dictionary `medals_dict` with the Olympic editions (years) as keys and DataFrames as values.
    b) The dictionary is built up inside a loop over the `year` of each Olympic edition (from the Index of `editions`).Once the dictionary of DataFrames is built up, you will combine the DataFrames using `pd.concat()`

INSTRUCTIONS

- Within the `for` loop:
  - Create the file path
  - Read `file_path` into a DataFrame. Assign the result to the `year`key of `medals_dict`.
  - Select only the columns `'Athlete'`, `'NOC'`, and `'Medal'` from `medals_dict[year]`.
  - Create a *new* column called `'Edition'` in the DataFrame `medals_dict[year]` whose entries are *all* `year`.
- Concatenate the dictionary of DataFrames `medals_dict` into a DataFame called `medals`. Specify the keyword argument `ignore_index=True` to prevent repeated integer indices.

  Print the first and last 5 rows of `medals`.

Counting medals by country/edition in a pivot table

1. Here, you'll start with the concatenated DataFrame `medals` from the previous exercise. You can construct a *pivot table* to see the number of medals each country won in each year. The result is a new DataFrame with the Olympic edition on the Index and with 138 country `NOC` codes as columns.

INSTRUCTIONS

- Construct a pivot table from the DataFrame `medals`, aggregating by `count` (by specifying the `aggfunc` parameter). Use `'Edition'` as the `Index`, `'Athlete'` for the `values`, and `'NOC'` for the `columns`.

Save the output by using following command in ex1d_eval.csv file **medal_counts.to_csv("/code/ex1d_eval.csv")**

Computing fraction of medals per Olympic edition

4) In this exercise, you'll start with the DataFrames `editions`, `medals`, & `medal_counts` from prior exercises.

   a) You can extract a Series with the total number of medals awarded in each Olympic edition.The DataFrame `medal_counts` can be divided row-wise by the total number of medals awarded each edition; the method `.divide()` performs the broadcast as you require.

INSTRUCTIONS

- Set the index of the DataFrame `editions` to be `'Edition'` (using the method `.set_index()`). Save the result as `totals`.
- Extract the `'Grand Total'` column from `totals` and assign the result back to `totals`.
- Divide the DataFrame `medal_counts` by `totals` along each row. You will have to use the `.divide()` method with the option `axis='rows'`. Assign the result to `fractions`.

Computing percentage change in fraction of medals won

5) Here, you'll start with the DataFrames `editions`, `medals`, `medal_counts`, & `fractions` from prior exercises.To see if there is a host country advantage, you first want to see how the fraction of medals won changes from edition to edition.

a) The *expanding mean* provides a way to see this down each column. It is the value of the mean with all the data available up to that point in time.

INSTRUCTIONS

- Create `mean_fractions` by chaining the methods `.expanding().mean()` to `fractions`.
- Compute the percentage change in `mean_fractions` down each column by applying `.pct_change()` and multiplying by `100`. Assign the result to `fractions_change`.
- Reset the index of `fractions_change` using the `.reset_index()` method. This will make `'Edition'` an ordinary column.
- Save the output by using following command in ex1f_eval.csv file **fractions_change.to_csv("/code/ex1f_eval.csv")**

Building hosts DataFrame

6) Your task here is to prepare a DataFrame `hosts` by left joining `editions` and `ioc_codes`. Once created, you will subset the `Edition` and `NOC` columns and set `Edition` as the Index. There are some missing `NOC` values; you will set those explicitly.Finally, you'll reset the Index & print the final DataFrame.

INSTRUCTIONS

- Create the DataFrame `hosts` by doing a *left join* on DataFrames `editions` and `ioc_codes` (using `pd.merge()`).
- Clean up `hosts` by subsetting and setting the Index.
    - Extract the columns `'Edition'` and `'NOC'`.
    - Set `'Edition'` column as the Index.
- Use the `.loc[]` accessor to find and assign the missing values to the `'NOC'` column in `hosts`. This has been done for you.
- Reset the index of `hosts` using `.reset_index`

Save the output by using following command in **ex1g_eval.csv** file
hosts.to_csv("/code/ex1g_eval.csv")

Reshaping for analysis

7) This exercise starts off with `fractions_change` and `hosts` .Your task here is to reshape the `fractions_change` DataFrame for later analysis. Initially, `fractions_change` is a wide DataFrame of 26 rows (one for each Olympic edition) and 139 columns (one for the edition and 138 for the competing countries).

   On reshaping with `pd.melt()`, as you will see, the result is a tall DataFrame with 3588 rows and 3 columns that summarizes the fractional change in the expanding mean of the percentage of medals won for each country in blocks.

INSTRUCTIONS

- Create a DataFrame `reshaped` by reshaping the DataFrame `fractions_change` with `pd.melt()`.
- You'll need to use the keyword argument `id_vars='Edition'` to set the identifier variable.
- You'll also need to use the keyword argument `value_name='Change'` to set the measured variables.
- Print the shape of the DataFrames `reshaped` and `fractions_change`. Create a DataFrame `chn` by extracting all the rows from `reshaped` in which the three letter code for each country (`'NOC'`) is `'CHN'`.
- Print the last 5 rows of the DataFrame `chn` using the `.tail()`method

Merging to compute influence

8) This exercise starts off with the DataFrames `reshaped` and `hosts` . Your task is to merge the two DataFrames and tidy the result. The end result is a DataFrame summarizing the fractional change in the expanding mean of the percentage of medals won for the *host country* in each Olympic edition.

INSTRUCTIONS

- Merge `reshaped` and `hosts` using an inner join. Remember, `how='inner'` is the default behavior for `pd.merge()`.
- Print the first 5 rows of the DataFrame `merged`. This has been done for you. You should see that the rows are jumbled chronologically.
- Set the index of `merged` to be `'Edition'` and sort the index.
- Save the output by using following command in **ex1i_eval.csv**
  influence.to_csv("/code/ex1i_eval.csv")

  .

Plotting influence of host country

This final exercise starts off with the DataFrames `influence` and `editions`.Your job is to plot the influence of being a host country.

INSTRUCTIONS

- Create a Series called `change` by extracting the `'Change'` column from `influence`.
- Create a bar plot of `change` using the `.plot()` method with `kind='bar'`. Save the result as `ax` to permit further customization.
- Customize the bar plot of `change` to improve readability:
- Apply the method `.set_ylabel("% Change of Host Country Medal Count")` to`ax`.
- Apply the method `.set_title("Is there a Host Country Advantage?")` to `ax`.
- Apply the method `.set_xticklabels(editions['City'])` to `ax`.
- Reveal the final plot using `plt.show()`.
- Save the following image using below command
  plt.savefig('/code/output/host_plot.png')