Exploring and pre-processing a dataset using Pandas

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Explore the dataset
- Pre-process dataset as required (may be for visualization)

Introduction

The aim of this lab is to provide you a refresher on the **Pandas** library, so that you can pre-process and anlyse the datasets before applying data visualization techniques on it. This lab will work as acrash course on *pandas*. if you are interested in learning more about the *pandas* library, detailed description and explanation of how to use it and how to clean, munge, and process data stored in a *pandas* dataframe are provided in other IBM courses.

Table of Contents

Exploring Datasets with *pandas*

pandas is an essential data analysis toolkit for Python. From their website:

> *pandas* is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive.

> It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python.

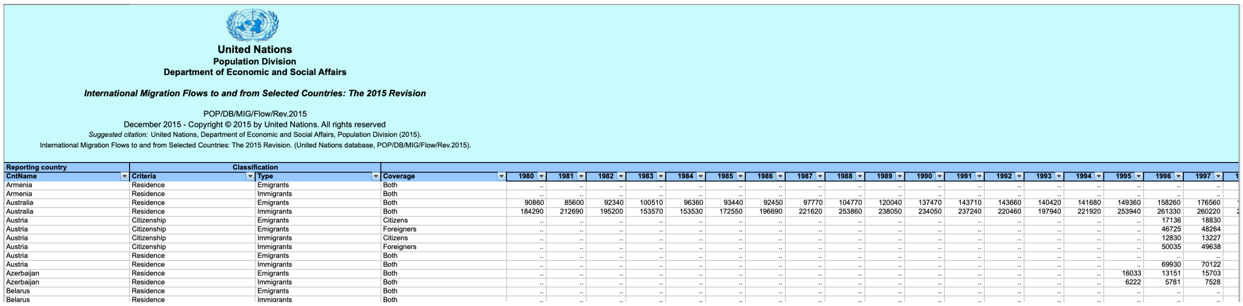
The course heavily relies on *pandas* for data wrangling, analysis, and visualization. We encourage you to spend some time and familiarize yourself with the *pandas* API Reference: http://pandas.pydata.org/pandas-docs/stable/api.html.

The Dataset: Immigration to Canada from 1980 to 2013

Dataset Source: International migration flows to and from selected countries - The 2015 revision.

The dataset contains annual data on the flows of international immigrants as recorded by the countries of destination. The data presents both inflows and outflows according to the place of birth, citizenship or place of previous / next residence both for foreigners and nationals. The current version presents data pertaining to 45 countries.

In this lab, we will focus on the Canadian immigration data.



United Nations
Population Division
Department of Economic and Social Affairs

International Migration Flows to and from Selected Countries: The 2015 Revision

POP/DB/MIG/Flow/Rev.2015
December 2015 - Copyright © 2015 by United Nations. All rights reserved
Suggested citation: United Nations, Department of Economic and Social Affairs, Population Division (2015).
International Migration Flows to and from Selected Countries: The 2015 Revision. (United Nations database, POP/DB/MIG/Flow/Rev.2015).

Reporting country	Classification																				
CntName	Criteria	Type	Coverage	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997
Armenia	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Armenia	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Australia	Residence	Emigrants	Both	90860	85600	92340	100510	96360	93440	92450	97770	104770	120040	137470	143710	143660	140420	141680	149360	158260	176560
Australia	Residence	Immigrants	Both	184290	212690	195200	153570	153530	172550	196690	221620	253860	238050	234050	237240	220460	197940	221920	253940	261330	260220
Austria	Citizenship	Emigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	17136	18830
Austria	Citizenship	Emigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	46725	48264
Austria	Citizenship	Immigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	12830	13227
Austria	Citizenship	Immigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	50035	49638
Austria	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Austria	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	69930	70122
Azerbaijan	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	16033	13151	15703
Azerbaijan	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	6222	5781	7528
Belarus	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Belarus	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..

The Canada Immigration dataset can be fetched from here.

---

pandas Basics

The first thing we'll do is install **openpyxl** (formerly **xlrd**), a module that *pandas* requires to read Excel files.

In [1]:

```
!mamba install openpyxl==3.0.9 -y
```

```
                  __    __    __    __
                 /  \  /  \  /  \  /  \
                /    \/    \/    \/    \
███████████████/  /██/  /██/  /██/  /████████████████████████
              /  / \   / \   / \   / \  \____
             /  /   \_/   \_/   \_/   \    o \__,
            / _/                       \_____/  `
            |/
        ███╗   ███╗ █████╗ ███╗   ███╗██████╗  █████╗
        ████╗ ████║██╔══██╗████╗ ████║██╔══██╗██╔══██╗
        ██╔████╔██║███████║██╔████╔██║██████╔╝███████║
        ██║╚██╔╝██║██╔══██║██║╚██╔╝██║██╔══██╗██╔══██║
        ██║ ╚═╝ ██║██║  ██║██║ ╚═╝ ██║██████╔╝██║  ██║
        ╚═╝     ╚═╝╚═╝  ╚═╝╚═╝     ╚═╝╚═════╝ ╚═╝  ╚═╝

        mamba (1.4.2) supported by @QuantStack

        GitHub:  https://github.com/mamba-org/mamba
        Twitter: https://twitter.com/QuantStack

█████████████████████████████████████████████████████████████


Looking for: ['openpyxl==3.0.9']

[+] 0.0s
[+] 0.1s
pkgs/main/linux-64 ━━━━━━━━━━━━ ━━━━━━━━━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.1s
pkgs/main/noarch   ━━━━━━━━━━━━ ━━━━━━━━━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.1s
pkgs/r/linux-64    ━━ ━━━━━━━━━━━━━━━ ━━━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.1s
pkgs/r/noarch      ━━━━━━━━━━━━━ ━━━━━━━━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.1s[+] 0.2s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━ ━━━━━━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.2s
pkgs/main/noarch   ━━━━━━━━━━━━━━ ━━━━━━━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.2s
pkgs/r/linux-64    ━━━━ ━━━━━━━━━━━━━━━ ━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.2s
pkgs/r/noarch      ━━━━━━━ ━━━━━━━━━━━━━━━━━   0.0 B /  ??.?MB @  ??.?MB/s  0.2s[+] 0.3s
pkgs/main/linux-64 ━ ━━━━━━━━━━━━━━━ ━━━━━━━ 176.1kB /  ??.?MB @ 683.2kB/s  0.3s
pkgs/main/noarch   ━━━━━━━━ ━━━━━━━━━━━━━━━━ 495.6kB /  ??.?MB @   1.9MB/s  0.3s
pkgs/r/linux-64    ━━━━━━ ━━━━━━━━━━━━━━━ ━━ 237.6kB /  ??.?MB @ 889.9kB/s  0.3s
pkgs/r/noarch      ━━━━━━━━━ ━━━━━━━━━━━━━━━ 385.0kB /  ??.?MB @   1.4MB/s  0.3spkgs/mai
n/noarch                                   875.2kB @   2.5MB/s  0.4s
[+] 0.4s
pkgs/main/linux-64 ━━ ━━━━━━━━━━━━━━━ ━━━━━━ 507.9kB /  ??.?MB @   1.5MB/s  0.4s
pkgs/r/linux-64    ━━━━━━━━ ━━━━━━━━━━━━━━━━ 401.4kB /  ??.?MB @   1.3MB/s  0.4s
pkgs/r/noarch      ━━━━━━━━━━━━ ━━━━━━━━━━━━ 786.4kB /  ??.?MB @   2.1MB/s  0.4s[+] 0.5s
pkgs/main/linux-64 ━━━━━ ━━━━━━━━━━━━━━━ ━━━ 995.3kB /  ??.?MB @   2.2MB/s  0.5s
pkgs/r/linux-64    ━━━━━━━━━━ ━━━━━━━━━━━━━━   1.1MB /  ??.?MB @   2.2MB/s  0.5s
pkgs/r/noarch      ━━━━━━━━━━━━━ ━━━━━━━━━━━   1.1MB /  ??.?MB @   2.4MB/s  0.5s[+] 0.6s
pkgs/main/linux-64 ━━━━━━━ ━━━━━━━━━━━━━━━ ━   1.4MB /  ??.?MB @   2.5MB/s  0.6s
pkgs/r/linux-64    ━━━━━━━━━━━━ ━━━━━━━━━━━━   1.5MB /  ??.?MB @   2.6MB/s  0.6s
pkgs/r/noarch      ━━━━━━━━━━━━━━━ ━━━━━━━━━   1.5MB /  ??.?MB @   2.7MB/s  0.6s[+] 0.7s
pkgs/main/linux-64 ━━━━━━━━━ ━━━━━━━━━━━━━━   1.8MB @   2.7MB/s             0.7s
pkgs/r/linux-64    ━━━━━━━━━━━━━━━━━━━━━━━━   1.9MB @   2.8MB/s Finalizing  0.7s
pkgs/r/noarch      ━ ━━━━━━━━━━━━━━━ ━━━━━━   1.9MB @   2.9MB/s             0.7spkgs/r/l
inux-64                                    @   2.8MB/s  0.7s
[+] 0.8s
pkgs/main/linux-64 ━━━━━━━━━━ ━━━━━━━━━━━━━   2.1MB @   2.8MB/s             0.8s
pkgs/r/noarch      ━━━━━━━━━━━━━━━━━━━━━━━━   2.3MB @   3.0MB/s Finalizing  0.8spkgs/r/n
oarch                                      @   3.0MB/s  0.8s
[+] 0.9s
pkgs/main/linux-64 ━━━━━━━━━━━━━ ━━━━━━━━━━━   2.7MB /  ??.?MB @   3.0MB/s  0.9s[+] 1.0s
pkgs/main/linux-64 ━━━━━━━━━━━━━━ ━━━━━━━━━━   2.9MB /  ??.?MB @   3.1MB/s  1.0s[+] 1.1s
pkgs/main/linux-64 ━━━━━━━━━ ━━━━━━━━━━━━━━━   3.3MB /  ??.?MB @   3.1MB/s  1.1s[+] 1.2s
pkgs/main/linux-64 ━━━━━━━━━━━ ━━━━━━━━━━━━━   3.6MB /  ??.?MB @   3.1MB/s  1.2s[+] 1.3s
pkgs/main/linux-64 ━━━━━━━━━━━━ ━━━━━━━━━━━━   3.8MB /  ??.?MB @   3.1MB/s  1.3s[+] 1.4s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━ ━━━━━━━━━   3.9MB /  ??.?MB @   2.8MB/s  1.4s[+] 1.5s
pkgs/main/linux-64 ━ ━━━━━━━━━━━━━━━ ━━━━━━━   4.2MB /  ??.?MB @   2.9MB/s  1.5s[+] 1.6s
pkgs/main/linux-64 ━━━ ━━━━━━━━━━━━━━━ ━━━━━   4.6MB /  ??.?MB @   3.0MB/s  1.6s[+] 1.7s
pkgs/main/linux-64 ━━━━━━ ━━━━━━━━━━━━━━━ ━━   5.1MB /  ??.?MB @   3.0MB/s  1.7s[+] 1.8s
pkgs/main/linux-64 ━━━━━━━━ ━━━━━━━━━━━━━━━━   5.5MB /  ??.?MB @   3.1MB/s  1.8s[+] 1.9s
pkgs/main/linux-64 ━━━━━━━━━━━ ━━━━━━━━━━━━━   5.9MB /  ??.?MB @   3.1MB/s  1.9s[+] 2.0s
pkgs/main/linux-64 ━━━━━━━━━━━━━ ━━━━━━━━━━━   6.3MB /  ??.?MB @   3.2MB/s  2.0s[+] 2.1s
```

```
pkgs/main/linux-64 ━━━━━━━ ━━━━━━━━━━━━━━━━   6.8MB /  ??.?MB @   3.2MB/s  2.1s[+] 2.2s
pkgs/main/linux-64 ━━━━━━━━━━ ━━━━━━━━━━━━━━   7.2MB /  ??.?MB @   3.3MB/s  2.2s[+] 2.3s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  2.3s[+] 2.4s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  2.4s[+] 2.5s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  2.5s[+] 2.6s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  2.6s[+] 2.7s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  2.7s[+] 2.8s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  2.8s[+] 2.9s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  2.9s[+] 3.0s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  3.0s[+] 3.1s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  3.1s[+] 3.2s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  3.2s[+] 3.3s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  3.3s[+] 3.4s
pkgs/main/linux-64 ━━━━━━━━━━━━━━━━━━━━━━━━   7.3MB @   3.3MB/s Finalizing  3.4spkgs/mai
n/linux-64                                @   3.3MB/s  3.4s

Pinned packages:
  - python 3.7.*


Transaction

  Prefix: /home/jupyterlab/conda/envs/python

  Updating specs:

   - openpyxl==3.0.9
   - ca-certificates
   - certifi
   - openssl


  Package               Version  Build            Channel                 Size
─────────────────────────────────────────────────────────────────────────────────
  Install:
─────────────────────────────────────────────────────────────────────────────────

  + et_xmlfile            1.1.0  py37h06a4308_0   pkgs/main/linux-64      10kB
  + openpyxl              3.0.9  pyhd3eb1b0_0     pkgs/main/noarch       168kB

  Upgrade:
─────────────────────────────────────────────────────────────────────────────────

  - ca-certificates  2023.5.7  hbcca054_0       conda-forge
  + ca-certificates  2024.7.2  h06a4308_0       pkgs/main/linux-64     130kB
  - openssl            1.1.1t  h0b41bf4_0       conda-forge
  + openssl            1.1.1w  h7f8727e_0       pkgs/main/linux-64       4MB

  Summary:

  Install: 2 packages
  Upgrade: 2 packages

  Total download: 4MB

─────────────────────────────────────────────────────────────────────────────────


[+] 0.0s
Downloading  (1) ━━━━━━━━━━━━━━━━━━━━━━━   0.0 B ca-certificates            0.0s
Extracting       ━━━━━━━━━━━━━━━━━━━━━━━       0                            0.0s[+] 0.1s
Downloading  (4) ━━━━━━━━━━━━━━━━━━━━━━━   0.0 B ca-certificates            0.1s
Extracting       ━━━━━━━━━━━━━━━━━━━━━━━       0                            0.0s[+] 0.2s
Downloading  (4) ━━━━━━━━━━━━━━━━━━━━━━━   0.0 B ca-certificates            0.2s
Extracting       ━━━━━━━━━━━━━━━━━━━━━━━       0                            0.0sca-certi
```

```
ficates                                    129.5kB @ 509.3kB/s  0.3s
et_xmlfile                                          10.2kB @  40.0kB/s  0.3s
openpyxl                                           167.6kB @ 653.1kB/s  0.3s
[+] 0.3s
Downloading  (1) ━━━━━━━━━━━━━━━━━━━━━━━ 524.0kB openssl                     0.3s
Extracting   (3) ━━━━━━━━━━━━━━━━━━━━━━━       0 ca-certificates            0.0sopenssl
3.9MB @  11.6MB/s  0.3s
[+] 0.4s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 ca-certificates            0.1s[+] 0.5s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 ca-certificates            0.2s[+] 0.6s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 ca-certificates            0.3s[+] 0.7s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 et_xmlfile                 0.4s[+] 0.8s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 et_xmlfile                 0.5s[+] 0.9s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 et_xmlfile                 0.6s[+] 1.0s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 et_xmlfile                 0.7s[+] 1.1s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openpyxl                   0.8s[+] 1.2s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openpyxl                   0.9s[+] 1.3s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openpyxl                   1.0s[+] 1.4s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openpyxl                   1.1s[+] 1.5s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openssl                    1.2s[+] 1.6s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openssl                    1.3s[+] 1.7s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openssl                    1.4s[+] 1.8s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 openssl                    1.5s[+] 1.9s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (4) ━━━━━━━━━━━━━━━━━━━━━━━       0 ca-certificates            1.6s[+] 2.0s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (3) ━━━━━━━━━━━━━━━━━━━━━━━       1 ca-certificates            1.7s[+] 2.1s
Downloading      ━━━━━━━━━━━━━━━━━━━━━━━   4.2MB                             0.4s
Extracting   (2) ━━━━━━━━━━━━━━━━━━━━━━━       2 openpyxl                   1.8s
Downloading and Extracting Packages

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Next, we'll do is import two key data analysis modules: *pandas* and *numpy*.

In [2]:

```python
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
```

Let's download and import our primary Canadian Immigration dataset using *pandas*'s `read_excel()` method.

In [3]:

```python
df_can = pd.read_excel(
    'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSki
    sheet_name='Canada by Citizenship',
    skiprows=range(20),
    skipfooter=2)

print('Data read into a pandas dataframe!')
```

```
Data read into a pandas dataframe!
```

Let's view the top 5 rows of the dataset using the `head()` function.

In [4]:

```
df_can.head()
# tip: You can specify the number of rows you'd like to see as follows: df_can.head(10)
```

Out[4]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	...	2
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	...	1
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	...	3
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	...	
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	...	

5 rows × 43 columns

We can also view the bottom 5 rows of the dataset using the `tail()` function.

In [5]:

```
df_can.tail()
```

Out[5]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2
190	Immigrants	Foreigners	Viet Nam	935	Asia	920	South-Eastern Asia	902	Developing regions	1191	...	
191	Immigrants	Foreigners	Western Sahara	903	Africa	912	Northern Africa	902	Developing regions	0	...	
192	Immigrants	Foreigners	Yemen	935	Asia	922	Western Asia	902	Developing regions	1	...	
193	Immigrants	Foreigners	Zambia	903	Africa	910	Eastern Africa	902	Developing regions	11	...	
194	Immigrants	Foreigners	Zimbabwe	903	Africa	910	Eastern Africa	902	Developing regions	72	...	

5 rows × 43 columns

When analyzing a dataset, it's always a good idea to start by getting basic information about your dataframe. We can do this by using the `info()` method.

This method can be used to get a short summary of the dataframe.

In [9]:

```
df_can.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Columns: 43 entries, Type to 2013
dtypes: int64(37), object(6)
memory usage: 65.6+ KB
```

To get the list of column headers we can call upon the data frame's `columns` instance variable.

In [10]:

```
df_can.columns
```

Out[10]:

```
Index([    'Type', 'Coverage',   'OdName',     'AREA', 'AreaName',      'REG',
        'RegName',      'DEV',  'DevName',       1980,       1981,       1982,
             1983,       1984,       1985,       1986,       1987,       1988,
             1989,       1990,       1991,       1992,       1993,       1994,
             1995,       1996,       1997,       1998,       1999,       2000,
             2001,       2002,       2003,       2004,       2005,       2006,
             2007,       2008,       2009,       2010,       2011,       2012,
             2013],
      dtype='object')
```

Similarly, to get the list of indices we use the `.index` instance variables.

In [12]:

```
df_can.index
```

Out[12]:

```
RangeIndex(start=0, stop=195, step=1)
```

Note: The default type of intance variables `index` and `columns` are **NOT** `list`.

In [13]:

```
print(type(df_can.columns))
print(type(df_can.index))
```

```
<class 'pandas.core.indexes.base.Index'>
<class 'pandas.core.indexes.range.RangeIndex'>
```

To get the index and columns as lists, we can use the `tolist()` method.

In [14]:

```
df_can.columns.tolist()
```

Out[14]:

```
['Type',
 'Coverage',
 'OdName',
 'AREA',
 'AreaName',
 'REG',
 'RegName',
 'DEV',
 'DevName',
 1980,
 1981,
 1982,
 1983,
 1984,
 1985,
 1986,
 1987,
 1988,
 1989,
 1990,
 1991,
 1992,
 1993,
 1994,
 1995,
 1996,
 1997,
 1998,
 1999,
 2000,
 2001,
 2002,
 2003,
 2004,
 2005,
 2006,
 2007,
 2008,
 2009,
 2010,
 2011,
 2012,
 2013]
```

In [15]:

```
df_can.index.tolist()
```

Out[15]:

```
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32,
 33,
 34,
 35,
 36,
 37,
 38,
 39,
 40,
 41,
 42,
 43,
 44,
 45,
 46,
 47,
 48,
 49,
 50,
 51,
 52,
 53,
 54,
 55,
 56,
 57,
 58,
 59,
 60,
 61,
 62,
 63,
 64,
 65,
 66,
 67,
 68,
 69,
 70,
 71,
 72,
 73,
 74,
 75,
 76,
 77,
 78,
 79,
 80,
 81,
 82,
 83,
 84,
 85,
 86,
 87,
 88,
 89,
 90,
 91,
 92,
 93,
 94,
 95,
 96,
 97,
 98,
          99,
          100,
          101,
          102,
          103,
          104,
          105,
          106,
          107,
          108,
          109,
          110,
          111,
          112,
          113,
          114,
          115,
          116,
          117,
          118,
          119,
          120,
          121,
          122,
          123,
          124,
          125,
          126,
          127,
          128,
          129,
          130,
          131,
          132,
          133,
          134,
          135,
          136,
          137,
          138,
          139,
          140,
          141,
          142,
          143,
          144,
          145,
          146,
          147,
          148,
          149,
          150,
          151,
          152,
          153,
          154,
          155,
          156,
          157,
          158,
          159,
          160,
          161,
          162,
          163,
 164,
 165,
 166,
 167,
 168,
 169,
 170,
 171,
 172,
 173,
 174,
 175,
 176,
 177,
 178,
 179,
 180,
 181,
 182,
 183,
 184,
 185,
 186,
 187,
 188,
 189,
 190,
 191,
 192,
 193,
 194]
```

In [16]:

```
print(type(df_can.columns.tolist()))
print(type(df_can.index.tolist()))
```

```
<class 'list'>
<class 'list'>
```

To view the dimensions of the dataframe, we use the `shape` instance variable of it.

In [17]:

```
# size of dataframe (rows, columns)
df_can.shape
```

Out[17]:

```
(195, 43)
```

Note: The main types stored in *pandas* objects are `float`, `int`, `bool`, `datetime64[ns]`, `datetime64[ns, tz]`, `timedelta[ns]`, `category`, and `object` (string). In addition, these dtypes have item sizes, e.g. `int64` and `int32`.

Let's clean the data set to remove a few unnecessary columns. We can use *pandas* `drop()` method as follows:

In [18]:

```
# in pandas axis=0 represents rows (default) and axis=1 represents columns.
df_can.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=True)
df_can.head(2)
```

Out[18]:

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	...	2004	2005
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	2978	3436
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	1450	1223

2 rows × 38 columns

Let's rename the columns so that they make sense. We can use `rename()` method by passing in a dictionary of old and new names as follows:

In [19]:

```
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'RegName':'Region'},
df_can.columns
```

Out[19]:

```
Index([  'Country', 'Continent',    'Region',   'DevName',        1980,
                1981,        1982,        1983,        1984,        1985,
                1986,        1987,        1988,        1989,        1990,
                1991,        1992,        1993,        1994,        1995,
                1996,        1997,        1998,        1999,        2000,
                2001,        2002,        2003,        2004,        2005,
                2006,        2007,        2008,        2009,        2010,
                2011,        2012,        2013],
      dtype='object')
```

We will also add a 'Total' column that sums up the total immigrants by country over the entire period 1980 - 2013, as follows:

In [21]:

```
df_can['Total'] = df_can.sum(axis=1)
df_can['Total']
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launcher.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=
None') is deprecated; in a future version this will raise TypeError.  Select only valid
columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

Out[21]:

```
0      117278
1       31398
2      138878
3          12
4          30
        ...  
190    194292
191         4
192      5970
193      3354
194     17196
Name: Total, Length: 195, dtype: int64
```

We can check to see how many null objects we have in the dataset as follows:

In [22]:

```
df_can.isnull().sum()
```

Out[22]:

```
Country      0
Continent    0
Region       0
```

```
DevName      0
1980         0
1981         0
1982         0
1983         0
1984         0
1985         0
1986         0
1987         0
1988         0
1989         0
1990         0
1991         0
1992         0
1993         0
1994         0
1995         0
1996         0
1997         0
1998         0
1999         0
2000         0
2001         0
2002         0
2003         0
2004         0
2005         0
2006         0
2007         0
2008         0
2009         0
2010         0
2011         0
2012         0
2013         0
Total        0
dtype: int64
```

Finally, let's view a quick summary of each column in our dataframe using the `describe()` method.

In [23]:

```
df_can.describe()
```

Out[23]:

	1980	1981	1982	1983	1984	1985	1986
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000
mean	508.394872	566.989744	534.723077	387.435897	376.497436	358.861538	441.271795
std	1949.588546	2152.643752	1866.997511	1204.333597	1198.246371	1079.309600	1225.576630
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.500000
50%	13.000000	10.000000	11.000000	12.000000	13.000000	17.000000	18.000000
75%	251.500000	295.500000	275.000000	173.000000	181.000000	197.000000	254.000000
max	22045.000000	24796.000000	20620.000000	10015.000000	10170.000000	9564.000000	9470.000000

8 rows × 35 columns

---

pandas Intermediate: Indexing and Selection (slicing)

Select Column

There are two ways to filter on a column name:

Method 1: Quick and easy, but only works if the column name does NOT have spaces or special characters.

```
df.column_name                # returns series
```

Method 2: More robust, and can filter on multiple columns.

```
df['column']                  # returns series
df[['column 1', 'column 2']]  # returns dataframe
```

Example: Let's try filtering on the list of countries ('Country').

In [24]:

```
df_can.Country  # returns a series
```

Out[24]:

```
0         Afghanistan
1             Albania
2             Algeria
3      American Samoa
4             Andorra
            ...
190          Viet Nam
191    Western Sahara
192             Yemen
193            Zambia
194          Zimbabwe
Name: Country, Length: 195, dtype: object
```

Let's try filtering on the list of countries ('Country') and the data for years: 1980 - 1985.

In [25]:

```
df_can[['Country', 1980, 1981, 1982, 1983, 1984, 1985]] # returns a dataframe
# notice that 'Country' is string, and the years are integers.
# for the sake of consistency, we will convert all column names to string later on.
```

Out[25]:

	Country	1980	1981	1982	1983	1984	1985
0	Afghanistan	16	39	39	47	71	340
1	Albania	1	0	0	0	0	0
2	Algeria	80	67	71	69	63	44
3	American Samoa	0	1	0	0	0	0
4	Andorra	0	0	0	0	0	0

	Country	1980	1981	1982	1983	1984	1985
...	...	...	...	...	...	...	...
190	Viet Nam	1191	1829	2162	3404	7583	5907
191	Western Sahara	0	0	0	0	0	0
192	Yemen	1	2	1	6	0	18
193	Zambia	11	17	11	7	16	9
194	Zimbabwe	72	114	102	44	32	29

195 rows × 7 columns

Select Row

There are main 2 ways to select rows:

```
df.loc[label]     # filters by the labels of the index/column
    df.iloc[index]    # filters by the positions of the index/column
```

Before we proceed, notice that the default index of the dataset is a numeric range from 0 to 194. This makes it very difficult to do a query by a specific country. For example to search for data on Japan, we need to know the corresponding index value.

This can be fixed very easily by setting the 'Country' column as the index using `set_index()` method.

In [26]:

```
df_can.set_index('Country', inplace=True)
# tip: The opposite of set is reset. So to reset the index, we can use df_can.reset_ind
```

In [27]:

```
df_can.head(3)
```

Out[27]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	20
Country													
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223	
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626	4

3 rows × 38 columns

In [28]:

```
# optional: to remove the name of the index
df_can.index.name = None
```

Example: Let's view the number of immigrants from Japan (row 87) for the following scenarios:

```
1. The full row data (all columns)
2. For year 2013
3. For years 1980 to 1985
```

In [29]:

```
# 1. the full row data (all columns)
df_can.loc['Japan']
```

Out[29]:

```
Continent                    Asia
Region               Eastern Asia
DevName      Developed regions
1980                          701
1981                          756
1982                          598
1983                          309
1984                          246
1985                          198
1986                          248
1987                          422
1988                          324
1989                          494
1990                          379
1991                          506
1992                          605
1993                          907
1994                          956
1995                          826
1996                          994
1997                          924
1998                          897
1999                         1083
2000                         1010
2001                         1092
2002                          806
2003                          817
2004                          973
2005                         1067
2006                         1212
2007                         1250
2008                         1284
2009                         1194
2010                         1168
2011                         1265
2012                         1214
2013                          982
Total                       55414
Name: Japan, dtype: object
```

In [30]:

```
# alternate methods
df_can.iloc[87]
```

Out[30]:

```
Continent                    Asia
Region               Eastern Asia
DevName      Developed regions
1980                          701
1981                          756
1982                          598
1983                          309
1984                          246
```

```
1985                      198
1986                      248
1987                      422
1988                      324
1989                      494
1990                      379
1991                      506
1992                      605
1993                      907
1994                      956
1995                      826
1996                      994
1997                      924
1998                      897
1999                     1083
2000                     1010
2001                     1092
2002                      806
2003                      817
2004                      973
2005                     1067
2006                     1212
2007                     1250
2008                     1284
2009                     1194
2010                     1168
2011                     1265
2012                     1214
2013                      982
Total                   55414
Name: Japan, dtype: object
```

In [31]:

```
df_can[df_can.index == 'Japan']
```

Out[31]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	200
Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198	248	...	1067	1212	125

1 rows × 38 columns

In [32]:

```
# 2. for year 2013
df_can.loc['Japan', 2013]
```

Out[32]:

```
982
```

In [33]:

```
# alternate method
# year 2013 is the last column, with a positional index of 36
df_can.iloc[87, 36]
```

Out[33]:

```
982
```

In [34]:

```
# 3. for years 1980 to 1985
df_can.loc['Japan', [1980, 1981, 1982, 1983, 1984, 1984]]
```

Out[34]:
```
1980    701
1981    756
1982    598
1983    309
1984    246
1984    246
Name: Japan, dtype: object
```

In [35]:
```
# Alternative Method
df_can.iloc[87, [3, 4, 5, 6, 7, 8]]
```

Out[35]:
```
1980    701
1981    756
1982    598
1983    309
1984    246
1985    198
Name: Japan, dtype: object
```

Exercise: Let's view the number of immigrants from **Haiti** for the following scenarios:

1. The full row data (all columns)
2. For year 2000
3. For years 1990 to 1995

In [36]:
```
df_can.loc['Haiti']
df_can.loc['Haiti', 2000]
df_can.loc['Haiti', [1990, 1991, 1992, 1993, 1994, 1995]]
```

Out[36]:
```
1990    2379
1991    2829
1992    2399
1993    3655
1994    2100
1995    2014
Name: Haiti, dtype: object
```

▶ Click here for a sample python solution

Column names that are integers (such as the years) might introduce some confusion. For example, when we are referencing the year 2013, one might confuse that when the 2013th positional index.

To avoid this ambuigity, let's convert the column names into strings: '1980' to '2013'.

In [37]:
```
df_can.columns = list(map(str, df_can.columns))
# [print (type(x)) for x in df_can.columns.values] #<-- uncomment to check type of colu
```

Since we converted the years to string, let's declare a variable that will allow us to easily call upon the full range of years:

In [38]:
```
# useful for plotting later on
years = list(map(str, range(1980, 2014)))
years
```

Out[38]:
```
['1980',
 '1981',
 '1982',
 '1983',
 '1984',
 '1985',
 '1986',
 '1987',
 '1988',
 '1989',
 '1990',
 '1991',
 '1992',
 '1993',
 '1994',
 '1995',
 '1996',
 '1997',
 '1998',
 '1999',
 '2000',
 '2001',
 '2002',
 '2003',
 '2004',
 '2005',
 '2006',
 '2007',
 '2008',
 '2009',
 '2010',
 '2011',
 '2012',
 '2013']
```

Exercise: Create a list named 'year' using map function for years ranging from 1990 to 2013. Then extract the data series from the dataframe df_can for Haiti using year list.

In [40]:

```python
year = list(map(str, range(1990, 2014)))
haiti = df_can.loc['Haiti', year] # passing in years 1990 - 2013
```

▶ Click here for a sample python solution

Filtering based on a criteria

To filter the dataframe based on a condition, we simply pass the condition as a boolean vector.

For example, Let's filter the dataframe to show the data on Asian countries (AreaName = Asia).

In [41]:

```python
# 1. create the condition boolean series
condition = df_can['Continent'] == 'Asia'
print(condition)
```

```
Afghanistan        True
Albania           False
Algeria           False
American Samoa    False
Andorra           False
                  ...
Viet Nam           True
Western Sahara    False
Yemen              True
Zambia            False
```

```
Zimbabwe          False
Name: Continent, Length: 195, dtype: bool
```

In [42]:

```
# 2. pass this condition into the dataFrame
df_can[condition]
```

Out[42]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436
Armenia	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	224
Azerbaijan	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	359
Bahrain	Asia	Western Asia	Developing regions	0	2	1	1	1	3	0	...	12
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	...	4171
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	0	...	5
Brunei Darussalam	Asia	South-Eastern Asia	Developing regions	79	6	8	2	2	4	12	...	4
Cambodia	Asia	South-Eastern Asia	Developing regions	12	19	26	33	10	7	8	...	370
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	...	42584
China, Hong Kong Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...	729
China, Macao Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...	21
Cyprus	Asia	Western Asia	Developing regions	132	128	84	46	46	43	48	...	7
Democratic People's Republic of Korea	Asia	Eastern Asia	Developing regions	1	1	3	1	4	3	0	...	14
Georgia	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	114
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36210

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005
Indonesia	Asia	South-Eastern Asia	Developing regions	186	178	252	115	123	100	127	...	632
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	...	5837
Iraq	Asia	Western Asia	Developing regions	262	245	260	380	428	231	265	...	2226
Israel	Asia	Western Asia	Developing regions	1403	1711	1334	541	446	680	1212	...	2446
Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198	248	...	1067
Jordan	Asia	Western Asia	Developing regions	177	160	155	113	102	179	181	...	1940
Kazakhstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	506
Kuwait	Asia	Western Asia	Developing regions	1	0	8	2	1	4	4	...	66
Kyrgyzstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	173
Lao People's Democratic Republic	Asia	South-Eastern Asia	Developing regions	11	6	16	16	7	17	21	...	42
Lebanon	Asia	Western Asia	Developing regions	1409	1119	1159	789	1253	1683	2576	...	3709
Malaysia	Asia	South-Eastern Asia	Developing regions	786	816	813	448	384	374	425	...	593
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	...	0
Mongolia	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...	59
Myanmar	Asia	South-Eastern Asia	Developing regions	80	62	46	31	41	23	18	...	210
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	13	...	607
Oman	Asia	Western Asia	Developing regions	0	0	0	8	0	0	0	...	14
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...	18139

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005
Qatar	Asia	Western Asia	Developing regions	0	0	0	0	0	0	1	...	11
Republic of Korea	Asia	Eastern Asia	Developing regions	1011	1456	1572	1081	847	962	1208	...	5832
Saudi Arabia	Asia	Western Asia	Developing regions	0	0	1	4	1	2	5	...	198
Singapore	Asia	South-Eastern Asia	Developing regions	241	301	337	169	128	139	205	...	392
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	...	4930
State of Palestine	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	453
Syrian Arab Republic	Asia	Western Asia	Developing regions	315	419	409	269	264	385	493	...	1458
Tajikistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	85
Thailand	Asia	South-Eastern Asia	Developing regions	56	53	113	65	82	66	78	...	575
Turkey	Asia	Western Asia	Developing regions	481	874	706	280	338	202	257	...	2065
Turkmenistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	40
United Arab Emirates	Asia	Western Asia	Developing regions	0	2	2	1	2	0	5	...	31
Uzbekistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	330
Viet Nam	Asia	South-Eastern Asia	Developing regions	1191	1829	2162	3404	7583	5907	2741	...	1852
Yemen	Asia	Western Asia	Developing regions	1	2	1	6	0	18	7	...	161

49 rows × 38 columns

In [43]:

```
# we can pass multiple criteria in the same line.
# let's filter for AreaNAme = Asia and RegName = Southern Asia

df_can[(df_can['Continent']=='Asia') & (df_can['Region']=='Southern Asia')]

# note: When using 'and' and 'or' operators, pandas requires we use '&' and '|' instead
# don't forget to enclose the two conditions in parentheses
```

Out[43]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	...	4171	
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	0	...	5	
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36210	3
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	...	5837	
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	...	0	
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	13	...	607	
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314	1
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	...	4930	

9 rows × 38 columns

Exercise: Fetch the data where AreaName is 'Africa' and RegName is 'Southern Africa'.
Display the dataframe and find out how many instances are there?

In [44]:

```
df_can[(df_can['Continent']=='Africa') & (df_can['Region']=='Southern Africa')]
```

Out[44]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	200
Botswana	Africa	Southern Africa	Developing regions	10	1	3	3	7	4	2	...	7	1
Lesotho	Africa	Southern Africa	Developing regions	1	1	1	2	7	5	3	...	4	
Namibia	Africa	Southern Africa	Developing regions	0	5	5	3	2	1	1	...	6	1
South Africa	Africa	Southern Africa	Developing regions	1026	1118	781	379	271	310	718	...	988	111
Swaziland	Africa	Southern Africa	Developing regions	4	1	1	0	10	7	1	...	7	

5 rows × 38 columns

▶ Click here for a sample python solution

Sorting Values of a Dataframe or Series

You can use the `sort_values()` function is used to sort a DataFrame or a Series based on one or more columns.
You to specify the column(s) by which you want to sort and the order (ascending or descending).
Below is the syntax to use it:-

```
 df.sort_values(col_name, axis=0, ascending=True, inplace=False,
ignore_index=False)
```

col_nam - the column(s) to sort by.
axis - axis along which to sort. 0 for sorting by rows (default) and 1 for sorting by columns.
ascending - to sort in ascending order (True, default) or descending order (False).
inplace - to perform the sorting operation in-place (True) or return a sorted copy (False, default).
ignore_index - to reset the index after sorting (True) or keep the original index values (False, default).

Let's sort out dataframe df_can on 'Total' column, in descending order to find out the top 5 countries that contributed the most to immigration to Canada.

In [48]:

```
df_can.sort_values(by='Total', ascending=False, axis=0, inplace=True)
top_5 = df_can.head(5)
top_5
```

Out[48]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36210
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	...	42584
United Kingdom of Great Britain and Northern Ireland	Europe	Northern Europe	Developed regions	22045	24796	20620	10015	10170	9564	9470	...	7258
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...	18139
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314

5 rows × 38 columns

Exercise: Find out top 3 countries that contributes the most to immigration to Canda in the year 2010.
Display the country names with the immigrant count in this year

In [57]:

```
df_can.sort_values(by='2010', ascending=False, axis=0, inplace=True)
top3_2010 = df_can['2010'].head(3)
top3_2010
```

Out[57]:

```
Philippines    38617
India          34235
China          30391
Name: 2010, dtype: int64
```

▶ Click here for a sample python solution

Congratulations! you have learned how to wrangle data with Pandas. You will be using alot of these commands to preprocess the data before its can be used for data visualization.

Thank you for completing this lab!

Author

Alex Aklson

Other Contributors

Jay Rajasekharan, Ehsan M. Kermani, Slobodan Markovic, Weiqing Wang, Dr. Pooja

<!-- --!>

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-06-08	2.5	Dr. Pooja	Separated from original lab
2021-05-29	2.4	Weiqing Wang	Fixed typos and code smells.
2021-01-20	2.3	Lakshmi Holla	Changed TOC cell markdown
2020-11-20	2.2	Lakshmi Holla	Changed IBM box URL
2020-11-03	2.1	Lakshmi Holla	Changed URL and info method
2020-08-27	2.0	Lavanya	Moved Lab to course repo in GitLab --!>

In [ ]: