Skills Network

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.

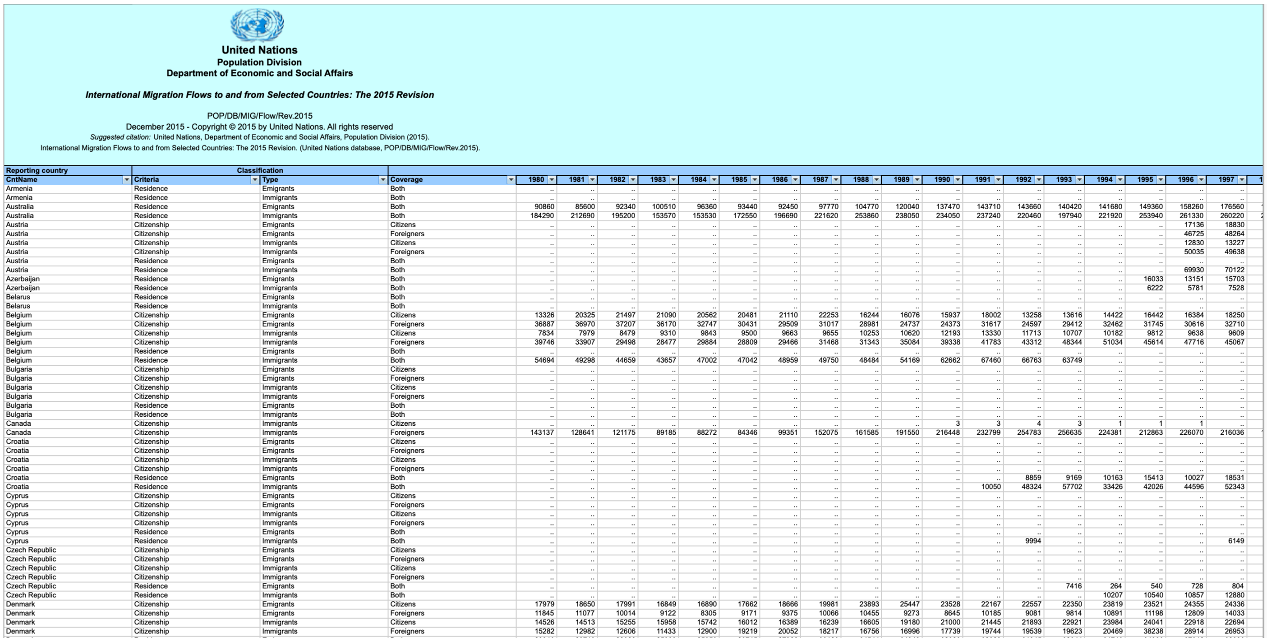
The course heavily relies on *pandas* for data wrangling, analysis, and visualization. We encourage you to spend some time and familiarize yourself with the *pandas* API Reference: http://pandas.pydata.org/pandas-docs/stable/api.html.

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	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Belgium	Citizenship	Emigrants	Citizens	13326	20325	21497	21090	20562	20481	21110	22253	16244	16076	15937	18002	13258	13616	14422	16442	16384	18250
Belgium	Citizenship	Emigrants	Foreigners	36887	36970	37207	36170	32747	30431	29509	31017	28981	24737	24373	31617	24597	29412	32462	31745	30616	32710
Belgium	Citizenship	Immigrants	Citizens	7834	7979	8479	9310	9843	9500	9663	9655	10253	10620	12193	13330	11713	10707	10182	9812	9638	9609
Belgium	Citizenship	Immigrants	Foreigners	39746	33907	29498	28477	29884	28809	29466	31468	31343	35084	39338	41783	43312	48344	51034	45614	47716	45067
Belgium	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Belgium	Residence	Immigrants	Both	54694	49298	44659	43657	47002	47042	48959	49750	48484	54169	62662	67460	66763	63749	..	..	..	..
Bulgaria	Citizenship	Emigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Bulgaria	Citizenship	Emigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Bulgaria	Citizenship	Immigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Bulgaria	Citizenship	Immigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Bulgaria	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Bulgaria	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Canada	Citizenship	Immigrants	Citizens	..	..	..	..	..	..	..	..	..	..	3	3	4	3	1	1	1	..
Canada	Citizenship	Immigrants	Foreigners	143137	128641	121175	89185	88272	84346	99351	152075	161585	191550	216448	232799	254783	256635	224381	212863	226070	216036
Croatia	Citizenship	Emigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Croatia	Citizenship	Emigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Croatia	Citizenship	Immigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Croatia	Citizenship	Immigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Croatia	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	8859	9169	10163	15413	10027	18531
Croatia	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	10050	48324	57702	33426	42026	44596	52343
Cyprus	Citizenship	Emigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Cyprus	Citizenship	Emigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Cyprus	Citizenship	Immigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Cyprus	Citizenship	Immigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Cyprus	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Cyprus	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	9994	..	..	..	..	6149
Czech Republic	Citizenship	Emigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Czech Republic	Citizenship	Emigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Czech Republic	Citizenship	Immigrants	Citizens	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Czech Republic	Citizenship	Immigrants	Foreigners	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
Czech Republic	Residence	Emigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	7416	264	540	728	804
Czech Republic	Residence	Immigrants	Both	..	..	..	..	..	..	..	..	..	..	..	..	..	..	10207	10540	10857	12880
Denmark	Citizenship	Emigrants	Citizens	17979	18650	17991	16849	16890	17662	18666	19981	23893	25447	23528	22167	22557	22350	23819	23521	24355	24336
Denmark	Citizenship	Emigrants	Foreigners	11845	11077	10014	9122	8305	9171	9375	10066	10455	9273	8645	10185	9081	9814	10891	11198	12809	14033
Denmark	Citizenship	Immigrants	Citizens	14526	14513	15255	15958	15742	16012	16389	16239	16605	19180	21000	21445	21893	22921	23984	24041	22918	22694
Denmark	Citizenship	Immigrants	Foreigners	15282	12982	12606	11433	12900	19219	20052	18217	16756	16996	17739	19744	19539	19623	20469	38238	28914	26953

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 [ ]: !mamba install openpyxl==3.0.9 -y
```

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

```
In [2]: 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]: df_can = pd.read_excel(
            'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.xlsx',
            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	...	2004	2005	2006	2007	2008	2009	2010	2011	2012
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	...	2978	3436	3009	2652	2111	1746	1758	2203	2635
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	...	1450	1223	856	702	560	716	561	539	620
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	...	3616	3626	4807	3623	4005	5393	4752	4325	3774
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	...	0	0	1	0	0	0	0	0	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	...	0	0	1	1	0	0	0	0	1

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	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2
190	Immigrants	Foreigners	Viet Nam	935	Asia	920	South-Eastern Asia	902	Developing regions	1191	...	1816	1852	3153	2574	1784	2171	1942	1723	1731	2
191	Immigrants	Foreigners	Western Sahara	903	Africa	912	Northern Africa	902	Developing regions	0	...	0	0	1	0	0	0	0	0	0	
192	Immigrants	Foreigners	Yemen	935	Asia	922	Western Asia	902	Developing regions	1	...	124	161	140	122	133	128	211	160	174	
193	Immigrants	Foreigners	Zambia	903	Africa	910	Eastern Africa	902	Developing regions	11	...	56	91	77	71	64	60	102	69	46	
194	Immigrants	Foreigners	Zimbabwe	903	Africa	910	Eastern Africa	902	Developing regions	72	...	1450	615	454	663	611	508	494	434	437	

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 [6]:
```
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 [7]:
```
df_can.columns
```

Out[7]:
```
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 [8]:
```
df_can.index
```

Out[8]:
```
RangeIndex(start=0, stop=195, step=1)
```

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

In [9]:
```
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 [ ]:
```
df_can.columns.tolist()
```

In [ ]:
```
df_can.index.tolist()
```

In [11]:
```
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 [12]:
```
# size of dataframe (rows, columns)
df_can.shape
```

Out[12]:
```
(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 [13]:
```
# 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[13]:

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	1450	1223	856	702	560	716	561	539	620	603

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 [14]:
```
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'RegName':'Region'}, inplace=True)
df_can.columns
```

Out[14]:
```
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 [15]:
```
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 reduct
ions (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[15]:
```
0      58639
1      15699
2      69439
3          6
4         15
       ...  
190    97146
191        2
192     2985
193     1677
194     8598
Name: Total, Length: 195, dtype: int64
```

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

In [ ]:
```
df_can.isnull().sum()
```

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

In [17]:
```
df_can.describe()
```

Out[17]:

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005
count	195.000000	195.000000	195.000000	195.000000	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	691.133333	714.389744	843.241026	...	1320.292308
std	1949.588546	2152.643752	1866.997511	1204.333597	1198.246371	1079.309600	1225.576630	2109.205607	2443.606788	2555.048874	...	4425.957828
min	0.000000	0.000000	0.000000	0.000000	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	0.500000	1.000000	1.000000	...	28.500000
50%	13.000000	10.000000	11.000000	12.000000	13.000000	17.000000	18.000000	26.000000	34.000000	44.000000	...	210.000000
75%	251.500000	295.500000	275.000000	173.000000	181.000000	197.000000	254.000000	434.000000	409.000000	508.500000	...	832.000000
max	22045.000000	24796.000000	20620.000000	10015.000000	10170.000000	9564.000000	9470.000000	21337.000000	27359.000000	23795.000000	...	42584.000000

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 [ ]:

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

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

In [19]:

```
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[19]:

	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
...	...	...	...	...	...	...	...
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 [20]:

```
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_index()
```

In [21]:

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

Out[21]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Country																					
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203	2635	2004	58639
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223	856	702	560	716	561	539	620	603	15699
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626	4807	3623	4005	5393	4752	4325	3774	4331	69439

3 rows × 38 columns

In [22]:

```
# 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 [24]:

```
# 1. the full row data (all columns)
df_can.loc['Japan'][:5]
```

Out[24]:

```
Continent                 Asia
Region            Eastern Asia
DevName       Developed regions
1980                       701
1981                       756
Name: Japan, dtype: object
```

In [26]:

```
# alternate methods
df_can.iloc[87][:5]
```

Out[26]:

```
Continent                 Asia
Region            Eastern Asia
DevName      Developed regions
1980                       701
1981                       756
Name: Japan, dtype: object
```

In [27]:

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

Out[27]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198	248	...	1067	1212	1250	1284	1194	1168	1265	1214	982	27707

1 rows × 38 columns

In [28]:

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

Out[28]:

```
982
```

In [29]:

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

Out[29]:

```
982
```

In [30]:

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

Out[30]:

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

In [31]:

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

Out[31]:

```
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 [32]:

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

Out[32]:

```
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 [33]:

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

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 [35]:

```
# useful for plotting later on
years = list(map(str, range(1980, 2014)))
print(years)
```

```
['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 [38]:

```
year = list(map(str, range(1990,2014)))
```

```
df_can[year].head()
```

Out[38]:

	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
Afghanistan	1028	1378	1170	713	858	1537	2212	2555	1999	2395	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
Albania	3	21	56	96	71	63	113	307	574	1264	...	1450	1223	856	702	560	716	561	539	620	603
Algeria	491	872	795	717	595	1106	2054	1842	2292	2389	...	3616	3626	4807	3623	4005	5393	4752	4325	3774	4331
American Samoa	2	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0
Andorra	3	0	1	0	0	0	0	0	2	0	...	0	0	1	1	0	0	0	0	1	1

5 rows × 24 columns

▸ 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 [39]:

```
# 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 [41]:

```
# 2. pass this condition into the dataFrame
df_can[condition][:5]
```

Out[41]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203	2635	2004	58639
Armenia	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	224	218	198	205	267	252	236	258	207	3310
Azerbaijan	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	359	236	203	125	165	209	138	161	57	2649
Bahrain	Asia	Western Asia	Developing regions	0	2	1	1	1	3	0	...	12	12	22	9	35	28	21	39	32	475
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	...	4171	4014	2897	2939	2104	4721	2694	2640	3789	65568

5 rows × 38 columns

In [42]:

```
# 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 of 'and' and 'or'
# don't forget to enclose the two conditions in parentheses
```

Out[42]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203	2635	2004	5
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	...	4171	4014	2897	2939	2104	4721	2694	2640	3789	6
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	0	...	5	10	7	36	865	1464	1879	1075	487	
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36210	33848	28742	28261	29456	34235	27509	30933	33087	69
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	...	5837	7480	6974	6475	6580	7477	7479	7534	11291	17
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	...	0	0	2	1	7	4	3	1	1	
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	13	...	607	540	511	581	561	1392	1129	1185	1308	1
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314	13127	10124	8994	7217	6811	7468	11227	12603	24
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	...	4930	4714	4123	4756	4547	4422	3309	3338	2394	14

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')][:5]
```

Out[44]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Botswana	Africa	Southern Africa	Developing regions	10	1	3	3	7	4	2	...	7	11	8	28	15	42	53	64	76	396
Lesotho	Africa	Southern Africa	Developing regions	1	1	1	2	7	5	3	...	4	0	4	1	8	7	1	0	6	107
Namibia	Africa	Southern Africa	Developing regions	0	5	5	3	2	1	1	...	6	19	13	26	14	16	23	24	83	320
South Africa	Africa	Southern Africa	Developing regions	1026	1118	781	379	271	310	718	...	988	1111	1200	1123	1188	1238	959	1243	1240	40568
Swaziland	Africa	Southern Africa	Developing regions	4	1	1	0	10	7	1	...	7	7	5	6	10	3	13	17	39	188

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 [45]:

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

Out[45]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36210	33848	28742	28261	29456	34235	27509	30933	33087
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	...	42584	33518	27642	30037	29622	30391	28502	33024	34129
United Kingdom of Great Britain and Northern Ireland	Europe	Northern Europe	Developed regions	22045	24796	20620	10015	10170	9564	9470	...	7258	7140	8216	8979	8876	8724	6204	6195	5827
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...	18139	18400	19837	24887	28573	38617	36765	34315	29544
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314	13127	10124	8994	7217	6811	7468	11227	12603

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 [47]:

```
df_can.sort_values(by='2010', ascending=False, axis=0, inplace=True)
top_3 = df_can['2010'].head(3)
top_3
```

Out[47]:

```
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 |--!> ##