Data Visualization

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Create Data Visualization with Python
- Use various Python libraries for visualization

Introduction

The aim of these labs is to introduce you to data visualization with Python as concrete and as consistent as possible. Speaking of consistency, because there is no *best* data visualization library avaiblable for Python - up to creating these labs - we have to introduce different libraries and show their benefits when we are discussing new visualization concepts. Doing so, we hope to make students well-rounded with visualization libraries and concepts so that they are able to judge and decide on the best visualitzation technique and tool for a given problem *and* audience.

Please make sure that you have completed the prerequisites for this course, namely **Python Basics for Data Science** and **Analyzing Data with Python**.

Note: The majority of the plots and visualizations will be generated using data stored in *pandas* dataframes. Therefore, in this lab, we provide a brief crash course on *pandas*. However, 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 our course **Analyzing Data with Python**.

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.

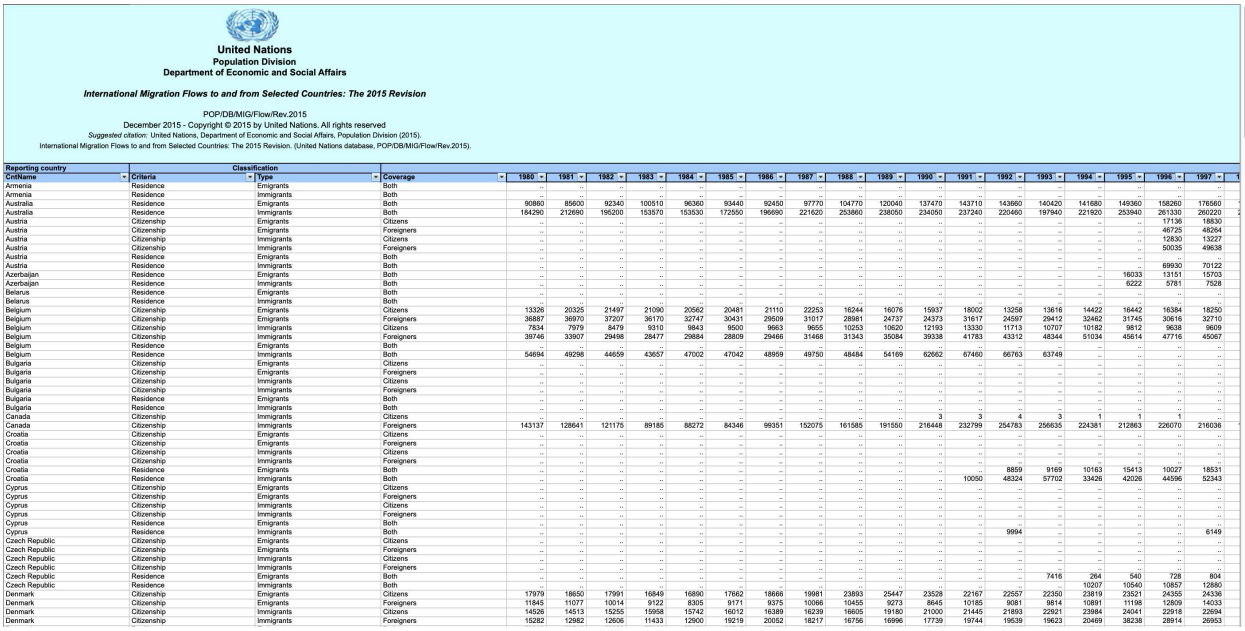
The course heavily relies on *pandas* for data wrangling, analysis, and visualization. We encourage you to spend some time and familizare yourself with the *pandas* API Reference: http://pandas.pydata.org/pandas-docs/stable/api.html.

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

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	5761	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	9858	10253	10620	12193	13330	11713	10707	10182	9612	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	254763	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	18868	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 import two key data analysis modules: *pandas* and **Numpy**.

In [1]:
```
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* `read_excel()` method. Normally, before we can do that, we would need to download a module which *pandas* requires to read in excel files. This module is **xlrd**. For your convenience, we have pre-installed this module, so you would not have to worry about that. Otherwise, you would need to run the following line of code to install the **xlrd** module:

```
!conda install -c anaconda xlrd --yes
```

Now we are ready to read in our data.

In [19]:
```
df_can = pd.read_excel('C:/Users/devro/Desktop/Jupyter/Data Visualization
with Python/Canada.xlsx',
                       sheet_name='Canada by Citizenship',
                       skiprows=range(20),
                       skipfooter=2,engine='openpyxl')
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 [20]:
```
df_can.head()
# tip: You can specify the number of rows you'd like to see as follows:
df_can.head(10)
```

Out[20]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	..
0	Immigrants	Foreigners	Afghanistan	935.0	Asia	5501.0	Southern Asia	902.0	Developing regions	16.0	..
1	Immigrants	Foreigners	Albania	908.0	Europe	925.0	Southern Europe	901.0	Developed regions	1.0	..
2	Immigrants	Foreigners	Algeria	903.0	Africa	912.0	Northern Africa	902.0	Developing regions	80.0	..
3	Immigrants	Foreigners	American Samoa	909.0	Oceania	957.0	Polynesia	902.0	Developing regions	0.0	..
4	Immigrants	Foreigners	Andorra	908.0	Europe	925.0	Southern Europe	901.0	Developed regions	0.0	..

5 rows × 51 columns

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

In [21]:
```
df_can.tail()
```

Out[21]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2012
1008	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
1009	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
1010	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
1011	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
1012	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN

5 rows × 51 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 [22]:
```
df_can.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1013 entries, 0 to 1012
Columns: 51 entries, Type to Unnamed: 50
dtypes: float64(45), object(6)
memory usage: 403.7+ KB
```

To get the list of column headers we can call upon the dataframe's `.columns` parameter.

In [23]:
```
df_can.columns.values
```

Out[23]:
```
array(['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, 'Unnamed: 43', 'Unnamed: 44',
       'Unnamed: 45', 'Unnamed: 46', 'Unnamed: 47', 'Unnamed: 48',
       'Unnamed: 49', 'Unnamed: 50'], dtype=object)
```

Similarly, to get the list of indicies we use the `.index` parameter.

In [24]:
```
df_can.index.values
```

Out[24]:
```
array([   0,    1,    2, ..., 1010, 1011, 1012], dtype=int64)
```

Note: The default type of index and columns is NOT list.

In [25]:
```
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 [26]:
```
df_can.columns.tolist()
df_can.index.tolist()

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` parameter.

In [27]:
```
# size of dataframe (rows, columns)
df_can.shape
```

Out[27]: (1013, 51)

Note: The main types stored in *pandas* objects are *float*, *int*, *bool*, *datetime64[ns]* and *datetime64[ns, tz] (in >= 0.17.0)*, *timedelta[ns]*, *category (in >= 0.15.0)*, 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 [28]:
```
# 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[28]:

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	...	2012	2
0	Afghanistan	Asia	Southern Asia	Developing regions	16.0	39.0	39.0	47.0	71.0	340.0	...	2635.0	2(
1	Albania	Europe	Southern Europe	Developed regions	1.0	0.0	0.0	0.0	0.0	0.0	...	620.0	6

2 rows × 46 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 [29]:
```
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent',
'RegName':'Region'}, inplace=True)
df_can.columns
```

Out[29]:
```
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, 'Unnamed: 43', 'Unnamed: 44',
       'Unnamed: 45', 'Unnamed: 46', 'Unnamed: 47', 'Unnamed: 48',
       'Unnamed: 49', 'Unnamed: 50'],
      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 [32]:
```
df_can['Total'] = df_can.sum(axis=1)
```

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

In [33]:
```
df_can.isnull().sum()
```

Out[33]:
```
Country         816
Continent       816
Region          816
DevName         816
1980            816
1981            816
1982            816
1983            816
1984            816
1985            816
1986            816
1987            816
1988            816
1989            816
1990            816
1991            816
1992            816
1993            816
1994            816
1995            816
1996            816
1997            816
1998            816
1999            816
2000            816
2001            816
2002            816
2003            816
2004            816
2005            816
2006            816
2007            816
2008            816
2009            816
2010            816
2011            816
2012            816
2013            816
Unnamed: 43    1013
```

```
Unnamed: 44    1013
Unnamed: 45    1013
Unnamed: 46    1013
Unnamed: 47    1013
Unnamed: 48    1013
Unnamed: 49    1013
Unnamed: 50    1013
Total             0
dtype: int64
```

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

In [34]:
```
df_can.describe()
```

Out[34]:

	1980	1981	1982	1983	1984	1985	
count	197.000000	197.000000	197.000000	197.000000	197.000000	197.000000	197.00
mean	1453.167513	1306.000000	1230.203046	905.431472	896.162437	856.304569	1008.63
std	10784.524807	9449.373841	8864.905615	6503.149859	6452.570413	6155.858422	7205.63
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.00
50%	14.000000	10.000000	12.000000	12.000000	14.000000	17.000000	21.00
75%	266.000000	299.000000	299.000000	197.000000	207.000000	202.000000	257.00
max	143137.000000	128641.000000	121175.000000	89185.000000	88272.000000	84346.000000	99351.00

8 rows × 43 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 [35]:
```
df_can.Country  # returns a series
```

Out[35]:
```
0          Afghanistan
1              Albania
2              Algeria
3       American Samoa
4              Andorra
              ...
1008               NaN
1009               NaN
1010               NaN
1011               NaN
1012               NaN
Name: Country, Length: 1013, dtype: object
```

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

In [36]:
```
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[36]:

	Country	1980	1981	1982	1983	1984	1985
0	Afghanistan	16.0	39.0	39.0	47.0	71.0	340.0
1	Albania	1.0	0.0	0.0	0.0	0.0	0.0
2	Algeria	80.0	67.0	71.0	69.0	63.0	44.0
3	American Samoa	0.0	1.0	0.0	0.0	0.0	0.0
4	Andorra	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
1008	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1009	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1010	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1011	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1012	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1013 rows × 7 columns

Select Row

There are main 3 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 defaul 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 corressponding index value.

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

In [37]:
```
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 [38]:
```
df_can.head(3)
```

Out[38]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2013
Country												
Afghanistan	Asia	Southern Asia	Developing regions	16.0	39.0	39.0	47.0	71.0	340.0	496.0	...	2004.0
Albania	Europe	Southern Europe	Developed regions	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	603.0
Algeria	Africa	Northern Africa	Developing regions	80.0	67.0	71.0	69.0	63.0	44.0	69.0	...	4331.0

3 rows × 46 columns

In [39]:
```
# 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 [40]:
```
# 1. the full row data (all columns)
print(df_can.loc['Japan'])

# alternate methods
print(df_can.iloc[87])
print(df_can[df_can.index == 'Japan'].T.squeeze())
```

```
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
Unnamed: 43                  NaN
Unnamed: 44                  NaN
Unnamed: 45                  NaN
Unnamed: 46                  NaN
Unnamed: 47                  NaN
Unnamed: 48                  NaN
Unnamed: 49                  NaN
Unnamed: 50                  NaN
Total                      83121
Name: Japan, dtype: object
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
Unnamed: 43                   NaN
Unnamed: 44                   NaN
Unnamed: 45                   NaN
Unnamed: 46                   NaN
Unnamed: 47                   NaN
Unnamed: 48                   NaN
Unnamed: 49                   NaN
Unnamed: 50                   NaN
Total                       83121
Name: Japan, dtype: object
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
Unnamed: 43                   NaN
Unnamed: 44                   NaN
Unnamed: 45                   NaN
Unnamed: 46                   NaN
Unnamed: 47                   NaN
Unnamed: 48                   NaN
Unnamed: 49                   NaN
Unnamed: 50                   NaN
Total                       83121
Name: Japan, dtype: object
```

In [41]:
```
# 2. for year 2013
print(df_can.loc['Japan', 2013])

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

```
982.0
982.0
```

In [42]:
```
# 3. for years 1980 to 1985
print(df_can.loc['Japan', [1980, 1981, 1982, 1983, 1984, 1984]])
print(df_can.iloc[87, [3, 4, 5, 6, 7, 8]])
```

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

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 [43]:
```
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 [44]:
```
# useful for plotting later on
```

```
years = list(map(str, range(1980, 2014)))
years
```

Out[44]:
```
['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']
```

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 [45]:
```
# 1. create the condition boolean series
condition = df_can['Continent'] == 'Asia'
print(condition)
```

```
Afghanistan        True
Albania           False
Algeria           False
American Samoa    False
Andorra           False
                  ...
NaN               False
NaN               False
NaN               False
NaN               False
NaN               False
Name: Continent, Length: 1013, dtype: bool
```

In [46]:
```
# 2. pass this condition into the dataFrame
df_can[condition]
```

Out[46]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986
Afghanistan	Asia	Southern Asia	Developing regions	16.0	39.0	39.0	47.0	71.0	340.0	496.0
Armenia	Asia	Western Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Azerbaijan	Asia	Western Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Bahrain	Asia	Western Asia	Developing regions	0.0	2.0	1.0	1.0	1.0	3.0	0.0
Bangladesh	Asia	Southern Asia	Developing regions	83.0	84.0	86.0	81.0	98.0	92.0	486.0
Bhutan	Asia	Southern Asia	Developing regions	0.0	0.0	0.0	0.0	1.0	0.0	0.0
Brunei Darussalam	Asia	South-Eastern Asia	Developing regions	79.0	6.0	8.0	2.0	2.0	4.0	12.0
Cambodia	Asia	South-Eastern Asia	Developing regions	12.0	19.0	26.0	33.0	10.0	7.0	8.0
China	Asia	Eastern Asia	Developing regions	5123.0	6682.0	3308.0	1863.0	1527.0	1816.0	1960.0
China, Hong Kong Special Administrative Region	Asia	Eastern Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
China, Macao Special Administrative Region	Asia	Eastern Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cyprus	Asia	Western Asia	Developing regions	132.0	128.0	84.0	46.0	46.0	43.0	48.0
Democratic People's Republic of Korea	Asia	Eastern Asia	Developing regions	1.0	1.0	3.0	1.0	4.0	3.0	0.0
Georgia	Asia	Western Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
India	Asia	Southern Asia	Developing regions	8880.0	8670.0	8147.0	7338.0	5704.0	4211.0	7150.0
Indonesia	Asia	South-Eastern Asia	Developing regions	186.0	178.0	252.0	115.0	123.0	100.0	127.0

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172.0	1429.0	1822.0	1592.0	1977.0	1648.0	1794.0
Iraq	Asia	Western Asia	Developing regions	262.0	245.0	260.0	380.0	428.0	231.0	265.0
Israel	Asia	Western Asia	Developing regions	1403.0	1711.0	1334.0	541.0	446.0	680.0	1212.0
Japan	Asia	Eastern Asia	Developed regions	701.0	756.0	598.0	309.0	246.0	198.0	248.0
Jordan	Asia	Western Asia	Developing regions	177.0	160.0	155.0	113.0	102.0	179.0	181.0
Kazakhstan	Asia	Central Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Kuwait	Asia	Western Asia	Developing regions	1.0	0.0	8.0	2.0	1.0	4.0	4.0
Kyrgyzstan	Asia	Central Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Lao People's Democratic Republic	Asia	South-Eastern Asia	Developing regions	11.0	6.0	16.0	16.0	7.0	17.0	21.0
Lebanon	Asia	Western Asia	Developing regions	1409.0	1119.0	1159.0	789.0	1253.0	1683.0	2576.0
Malaysia	Asia	South-Eastern Asia	Developing regions	786.0	816.0	813.0	448.0	384.0	374.0	425.0
Maldives	Asia	Southern Asia	Developing regions	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Mongolia	Asia	Eastern Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Myanmar	Asia	South-Eastern Asia	Developing regions	80.0	62.0	46.0	31.0	41.0	23.0	18.0
Nepal	Asia	Southern Asia	Developing regions	1.0	1.0	6.0	1.0	2.0	4.0	13.0
Oman	Asia	Western Asia	Developing regions	0.0	0.0	0.0	8.0	0.0	0.0	0.0
Pakistan	Asia	Southern Asia	Developing regions	978.0	972.0	1201.0	900.0	668.0	514.0	691.0
Philippines	Asia	South-Eastern Asia	Developing regions	6051.0	5921.0	5249.0	4562.0	3801.0	3150.0	4166.0
Qatar	Asia	Western Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	1.0

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986
Republic of Korea	Asia	Eastern Asia	Developing regions	1011.0	1456.0	1572.0	1081.0	847.0	962.0	1208.0
Saudi Arabia	Asia	Western Asia	Developing regions	0.0	0.0	1.0	4.0	1.0	2.0	5.0
Singapore	Asia	South-Eastern Asia	Developing regions	241.0	301.0	337.0	169.0	128.0	139.0	205.0
Sri Lanka	Asia	Southern Asia	Developing regions	185.0	371.0	290.0	197.0	1086.0	845.0	1838.0
State of Palestine	Asia	Western Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Syrian Arab Republic	Asia	Western Asia	Developing regions	315.0	419.0	409.0	269.0	264.0	385.0	493.0
Tajikistan	Asia	Central Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Thailand	Asia	South-Eastern Asia	Developing regions	56.0	53.0	113.0	65.0	82.0	66.0	78.0
Turkey	Asia	Western Asia	Developing regions	481.0	874.0	706.0	280.0	338.0	202.0	257.0
Turkmenistan	Asia	Central Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
United Arab Emirates	Asia	Western Asia	Developing regions	0.0	2.0	2.0	1.0	2.0	0.0	5.0
Uzbekistan	Asia	Central Asia	Developing regions	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Viet Nam	Asia	South-Eastern Asia	Developing regions	1191.0	1829.0	2162.0	3404.0	7583.0	5907.0	2741.0
Yemen	Asia	Western Asia	Developing regions	1.0	2.0	1.0	6.0	0.0	18.0	7.0

49 rows × 46 columns

In [47]:
```
# we can pass mutliple 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[47]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...
Afghanistan	Asia	Southern Asia	Developing regions	16.0	39.0	39.0	47.0	71.0	340.0	496.0	...
Bangladesh	Asia	Southern Asia	Developing regions	83.0	84.0	86.0	81.0	98.0	92.0	486.0	...
Bhutan	Asia	Southern Asia	Developing regions	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...
India	Asia	Southern Asia	Developing regions	8880.0	8670.0	8147.0	7338.0	5704.0	4211.0	7150.0	...
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172.0	1429.0	1822.0	1592.0	1977.0	1648.0	1794.0	...
Maldives	Asia	Southern Asia	Developing regions	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...
Nepal	Asia	Southern Asia	Developing regions	1.0	1.0	6.0	1.0	2.0	4.0	13.0	...
Pakistan	Asia	Southern Asia	Developing regions	978.0	972.0	1201.0	900.0	668.0	514.0	691.0	...
Sri Lanka	Asia	Southern Asia	Developing regions	185.0	371.0	290.0	197.0	1086.0	845.0	1838.0	...

9 rows × 46 columns

Before we proceed: let's review the changes we have made to our dataframe.

In [48]:
```
print('data dimensions:', df_can.shape)
print(df_can.columns)
df_can.head(2)
```

```
data dimensions: (1013, 46)
Index(['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', 'Unnamed: 43', 'Unnamed: 44', 'Unnamed: 45',
       'Unnamed: 46', 'Unnamed: 47', 'Unnamed: 48', 'Unnamed: 49',
       'Unnamed: 50', 'Total'],
      dtype='object')
```

Out[48]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2013
Afghanistan	Asia	Southern Asia	Developing regions	16.0	39.0	39.0	47.0	71.0	340.0	496.0	...	2004.0
Albania	Europe	Southern Europe	Developed regions	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	603.0

2 rows × 46 columns

Visualizing Data using Matplotlib

Matplotlib: Standard Python Visualization Library

The primary plotting library we will explore in the course is Matplotlib. As mentioned on their website:

> Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

If you are aspiring to create impactful visualization with python, Matplotlib is an essential tool to have at your disposal.

Matplotlib.Pyplot

One of the core aspects of Matplotlib is `matplotlib.pyplot` . It is Matplotlib's scripting layer which we studied in details in the videos about Matplotlib. Recall that it is a collection of command style functions that make Matplotlib work like MATLAB. Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In this lab, we will work with the scripting layer to learn how to generate line plots. In future labs, we will get to work with the Artist layer as well to experiment first hand how it differs from the scripting layer.

Let's start by importing `Matplotlib` and `Matplotlib.pyplot` as follows:

In [49]:
```
# we are using the inline backend
%matplotlib inline

import matplotlib as mpl
import matplotlib.pyplot as plt
```

*optional: check if Matplotlib is loaded.

In [50]:
```
print ('Matplotlib version: ', mpl.__version__) # >= 2.0.0
```

```
Matplotlib version:  3.3.3
```

*optional: apply a style to Matplotlib.

In [51]:
```
print(plt.style.available)
mpl.style.use(['ggplot']) # optional: for ggplot-like style
```

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast',
'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorbli
nd', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seabor
n-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seab
orn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind1
0']
```

Plotting in *pandas*

Fortunately, pandas has a built-in implementation of Matplotlib that we can use. Plotting in *pandas* is as simple as appending a `.plot()` method to a series or dataframe.

Documentation:

- Plotting with Series
- Plotting with Dataframes

Line Pots (Series/Dataframe)

What is a line plot and why use it?

A line chart or line plot is a type of plot which displays information as a series of data points called 'markers' connected by straight line segments. It is a basic type of chart common in many fields. Use line plot when you have a continuous data set. These are best suited for trend-based visualizations of data over a period of time.

Let's start with a case study:

In 2010, Haiti suffered a catastrophic magnitude 7.0 earthquake. The quake caused widespread devastation and loss of life and aout three million people were affected by this natural disaster. As part of Canada's humanitarian effort, the Government of Canada stepped up its effort in accepting refugees from Haiti. We can quickly visualize this effort using a `Line` plot:

Question: Plot a line graph of immigration from Haiti using `df.plot()`.

First, we will extract the data series for Haiti.

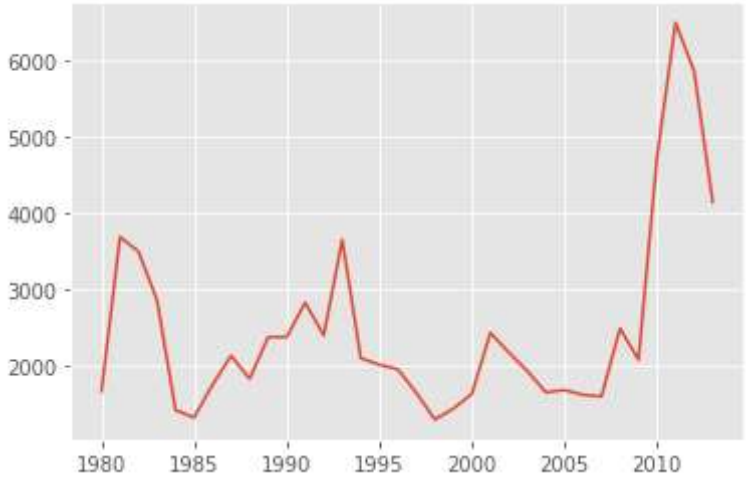
In [52]:
```
haiti = df_can.loc['Haiti', years] # passing in years 1980 - 2013 to
exclude the 'total' column
haiti.head()
```

Out[52]:
```
1980    1666
1981    3692
1982    3498
1983    2860
1984    1418
Name: Haiti, dtype: object
```

Next, we will plot a line plot by appending `.plot()` to the `haiti` dataframe.

In [53]:
```
haiti.plot()
```

Out[53]: <AxesSubplot:>



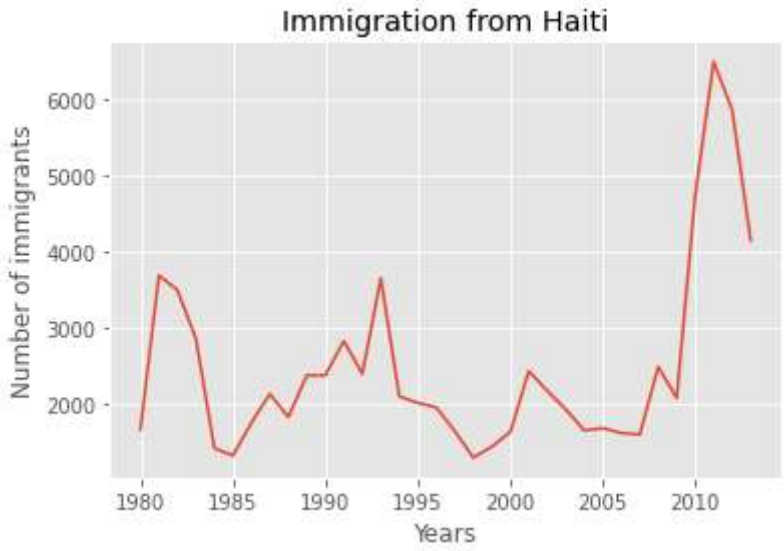
pandas automatically populated the x-axis with the index values (years), and the y-axis with the column values (population). However, notice how the years were not displayed because they are of type *string*. Therefore, let's change the type of the index values to *integer* for plotting.

Also, let's label the x and y axis using `plt.title()`, `plt.ylabel()`, and `plt.xlabel()` as follows:

In [54]:
```
haiti.index = haiti.index.map(int) # let's change the index values of Haiti to type integer for plotting
haiti.plot(kind='line')

plt.title('Immigration from Haiti')
plt.ylabel('Number of immigrants')
plt.xlabel('Years')

plt.show() # need this line to show the updates made to the figure
```



We can clearly notice how number of immigrants from Haiti spiked up from 2010 as Canada

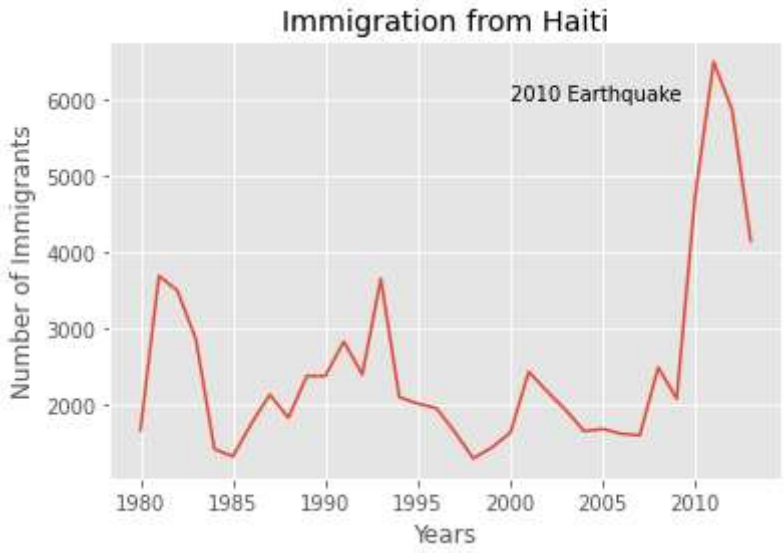
stepped up its efforts to accept refugees from Haiti. Let's annotate this spike in the plot by using the `plt.text()` method.

In [55]:
```
haiti.plot(kind='line')

plt.title('Immigration from Haiti')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')

# annotate the 2010 Earthquake.
# syntax: plt.text(x, y, label)
plt.text(2000, 6000, '2010 Earthquake') # see note below

plt.show()
```



With just a few lines of code, you were able to quickly identify and visualize the spike in immigration!

Quick note on x and y values in `plt.text(x, y, label)`:

> Since the x-axis (years) is type 'integer', we specified x as a year. The y axis (number of immigrants) is type 'integer', so we can just specify the value y = 6000.
>
> ```
> plt.text(2000, 6000, '2010 Earthquake') # years stored as type int
> ```
>
> If the years were stored as type 'string', we would need to specify x as the index position of the year. Eg 20th index is year 2000 since it is the 20th year with a base year of 1980.

```
plt.text(20, 6000, '2010 Earthquake') # years stored as type int
```

We will cover advanced annotation methods in later modules.

We can easily add more countries to line plot to make meaningful comparisons immigration from different countries.

Question: Let's compare the number of immigrants from India and China from 1980 to 2013.

Step 1: Get the data set for China and India, and display dataframe.

In [ ]:
```
### type your answer here
```

▶ Click here for a sample python solution

Step 2: Plot graph. We will explicitly specify line plot by passing in `kind` parameter to `plot()`.

In [ ]:
```
### type your answer here
```

▶ Click here for a sample python solution

That doesn't look right...

Recall that *pandas* plots the indices on the x-axis and the columns as individual lines on the y-axis. Since `df_CI` is a dataframe with the `country` as the index and `years` as the columns, we must first transpose the dataframe using `transpose()` method to swap the row and columns.

In [ ]:
```
df_CI = df_CI.transpose()
df_CI.head()
```

pandas will auomatically graph the two countries on the same graph. Go ahead and plot the new transposed dataframe. Make sure to add a title to the plot and label the axes.

In [ ]:
```
### type your answer here
```

▶ Click here for a sample python solution

From the above plot, we can observe that the China and India have very similar immigration trends through the years.

Note: How come we didn't need to transpose Haiti's dataframe before plotting (like we did for df_CI)?

That's because `haiti` is a series as opposed to a dataframe, and has the years as its indices as shown below.

```
print(type(haiti))
print(haiti.head(5))
```

```
class 'pandas.core.series.Series'
1980 1666
1981 3692
1982 3498
1983 2860
1984 1418
Name: Haiti, dtype: int64
```

Line plot is a handy tool to display several dependent variables against one independent variable. However, it is recommended that no more than 5-10 lines on a single graph; any more than that and it becomes difficult to interpret.

Question: Compare the trend of top 5 countries that contributed the most to immigration to Canada.

In [ ]:
```
### type your answer here
```

▶ Click here for a sample python solution

Other Plots

Congratulations! you have learned how to wrangle data with python and create a line plot with Matplotlib. There are many other plotting styles available other than the default Line plot, all of which can be accessed by passing `kind` keyword to `plot()`. The full list of available plots are as follows:

- `bar` for vertical bar plots
- `barh` for horizontal bar plots
- `hist` for histogram
- `box` for boxplot
- `kde` or `density` for density plots
- `area` for area plots
- `pie` for pie plots
- `scatter` for scatter plots
- `hexbin` for hexbin plot

Thank you for completing this lab!

Author

Alex Aklson

Other Contributors

Jay Rajasekharan Ehsan M. Kermani Slobodan Markovic.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
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