



(<https://cognitiveclass.ai>).

Introduction to Matplotlib and Line Plots

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 available 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 visualization technique and tool for a given problem *and* audience.

Please make sure that you have completed the prerequisites for this course, namely **[Python for Data Science](http://cocl.us/PY0101EN_DV0101EN_LAB1_Coursera)** (http://cocl.us/PY0101EN_DV0101EN_LAB1_Coursera), which is part of this specialization.

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 **[Data Analysis with Python](http://cocl.us/DA0101EN_DV0101EN_LAB1_Coursera)** (http://cocl.us/DA0101EN_DV0101EN_LAB1_Coursera), which is also part of this specialization.

Table of Contents

1. [Exploring Datasets with *pandas*](#)
 - 1.1 [The Dataset: Immigration to Canada from 1980 to 2013](#)
 - 1.2 [pandas Basics](#)
 - 1.3 [pandas Intermediate: Indexing and Selection](#)
 2. [Visualizing Data using Matplotlib](#)
 - 2.1 [Matplotlib: Standard Python Visualization Library](#)
 3. [Line Plots](#)
-

Exploring Datasets with *pandas*

pandas is an essential data analysis toolkit for Python. From their [website \(http://pandas.pydata.org/\)](http://pandas.pydata.org/):

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> (<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](http://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.shtml)
(<http://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.shtml>).

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.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	 <p style="text-align: center;">United Nations Population Division Department of Economic and Social Affairs</p> <p style="text-align: center;"><i>International Migration Flows to and from Selected Countries: The 2015 Revision</i></p> <p style="text-align: center;">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).</p>													
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17	Reporting country: Canada													
18	Criterion: Citizenship													
19														
20	Classification		Origin/Destination	Major area		Region		Development region						
21	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	1981	1982	1983	1984
22	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asi	902	Developing re	16	39	39	47	71
23	Immigrants	Foreigners	Albania	908	Europe	925	Southern Eur	901	Developed re	1	0	0	0	0
24	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Afric	902	Developing re	80	67	71	69	63
25	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing re	0	1	0	0	0
26	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Eur	901	Developed re	0	0	0	0	0
27	Immigrants	Foreigners	Angola	903	Africa	911	Middle Africa	902	Developing re	1	3	6	6	4

For sake of simplicity, Canada's immigration data has been extracted and uploaded to one of IBM servers.

You can fetch the data from [here](https://ibm.box.com/shared/static/lw190pt9zpy5bd1ptyg2aw15awomz9pu.xlsx)

(<https://ibm.box.com/shared/static/lw190pt9zpy5bd1ptyg2aw15awomz9pu.xlsx>).

pandas Basics

The first thing we'll do is import two key data analysis modules: *pandas* and **Numpy**.

In [27]:

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 [28]:

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

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

In [29]:

Out[29]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	198
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	1
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	8
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	

5 rows × 43 columns

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

In [30]:

Out[30]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	19
190	Immigrants	Foreigners	Viet Nam	935	Asia	920	South-Eastern Asia	902	Developing regions	11
191	Immigrants	Foreigners	Western Sahara	903	Africa	912	Northern Africa	902	Developing regions	
192	Immigrants	Foreigners	Yemen	935	Asia	922	Western Asia	902	Developing regions	
193	Immigrants	Foreigners	Zambia	903	Africa	910	Eastern Africa	902	Developing regions	
194	Immigrants	Foreigners	Zimbabwe	903	Africa	910	Eastern Africa	902	Developing regions	

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.

In [31]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 43 columns):
Type          195 non-null object
Coverage      195 non-null object
OdName        195 non-null object
AREA          195 non-null int64
AreaName      195 non-null object
REG           195 non-null int64
RegName       195 non-null object
DEV           195 non-null int64
DevName       195 non-null object
1980          195 non-null int64
1981          195 non-null int64
1982          195 non-null int64
1983          195 non-null int64
1984          195 non-null int64
1985          195 non-null int64
1986          195 non-null int64
1987          195 non-null int64
1988          195 non-null int64
1989          195 non-null int64
1990          195 non-null int64
1991          195 non-null int64
1992          195 non-null int64
1993          195 non-null int64
1994          195 non-null int64
1995          195 non-null int64
1996          195 non-null int64
1997          195 non-null int64
1998          195 non-null int64
1999          195 non-null int64
2000          195 non-null int64
2001          195 non-null int64
2002          195 non-null int64
2003          195 non-null int64
2004          195 non-null int64
2005          195 non-null int64
2006          195 non-null int64
2007          195 non-null int64
2008          195 non-null int64
2009          195 non-null int64
2010          195 non-null int64
2011          195 non-null int64
2012          195 non-null int64
2013          195 non-null int64
dtypes: int64(37), object(6)
memory usage: 65.6+ KB
```

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

In [32]:

```
Out[32]: 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], dtype=object)
```

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

In [33]:

```
Out[33]: array([ 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])
```

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

In [19]:

```
<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 [6]:

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

To view the dimensions of the dataframe, we use the `.shape` parameter.

In [34]:

Out[34]: (195, 43)

Note: The main types stored in *pandas* objects are *float*, *int*, *bool*, *datetime64[ns]* and *datetime64[ns, tz]* (in $\geq 0.17.0$), *timedelta[ns]*, *category* (in $\geq 0.15.0$), 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 [35]:

Out[35]:

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

2 rows x 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 [36]:

```
Out[36]: Index([  'Country', 'Continent',      'Region',      'DevName',      198
0,
          1981,      1982,      1983,      1984,      198
5,
          1986,      1987,      1988,      1989,      199
0,
          1991,      1992,      1993,      1994,      199
5,
          1996,      1997,      1998,      1999,      200
0,
          2001,      2002,      2003,      2004,      200
5,
          2006,      2007,      2008,      2009,      201
0,
          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 [37]:

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

In [20]:

```
Out[20]: 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
```

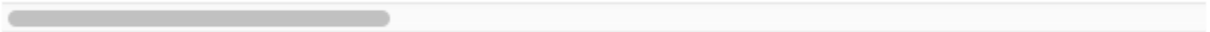
Finally, let's view a quick summary of each column in our dataframe using the `describe()` method.

In [21]:

Out[21]:

	1980	1981	1982	1983	1984	1985	
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195
mean	508.394872	566.989744	534.723077	387.435897	376.497436	358.861538	408
std	1949.588546	2152.643752	1866.997511	1204.333597	1198.246371	1079.309600	1204
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	13.000000	10.000000	11.000000	12.000000	13.000000	17.000000	13
75%	251.500000	295.500000	275.000000	173.000000	181.000000	197.000000	251
max	22045.000000	24796.000000	20620.000000	10015.000000	10170.000000	9564.000000	945

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 [38]:

```

Out[38]: 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
          ...
        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

```

Afghanistan
 Albania
 Algeria
 American Samoa
 Andorra
 Angola
 Antigua and Barbuda
 Argentina
 Armenia
 Australia
 Austria
 Azerbaijan
 Bahamas
 Bahrain
 Bangladesh
 Barbados
 Belarus
 Belgium
 Belize
 Benin
 Bhutan
 Bolivia (Plurinational State of)
 Bosnia and Herzegovina
 Botswana
 Brazil
 Brunei Darussalam
 Bulgaria
 Burkina Faso
 Burundi
 Cabo Verde
 ...
 Suriname
 Swaziland
 Sweden
 Switzerland
 Syrian Arab Republic
 Tajikistan
 Thailand
 The former Yugoslav Republic of Macedonia
 Togo
 Tonga
 Trinidad and Tobago
 Tunisia
 Turkey
 Turkmenistan
 Tuvalu
 Uganda
 Ukraine
 United Arab Emirates
 United Kingdom of Great Britain and Northern I...
 United Republic of Tanzania
 United States of America
 Uruguay
 Uzbekistan
 Vanuatu
 Venezuela (Bolivarian Republic of)
 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 ('OdName') and the data for years: 1980 - 1985.

In [39]:

Out[39]:

	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
5	Angola	1	3	6	6	4	3
6	Antigua and Barbuda	0	0	0	0	42	52
7	Argentina	368	426	626	241	237	196
8	Armenia	0	0	0	0	0	0
9	Australia	702	639	484	317	317	319
10	Austria	234	238	201	117	127	165
11	Azerbaijan	0	0	0	0	0	0
12	Bahamas	26	23	38	12	21	28
13	Bahrain	0	2	1	1	1	3
14	Bangladesh	83	84	86	81	98	92
15	Barbados	372	376	299	244	265	285
16	Belarus	0	0	0	0	0	0
17	Belgium	511	540	519	297	183	181
18	Belize	16	27	13	21	37	26
19	Benin	2	5	4	3	4	3
20	Bhutan	0	0	0	0	1	0
21	Bolivia (Plurinational State of)	44	52	42	49	38	44
22	Bosnia and Herzegovina	0	0	0	0	0	0
23	Botswana	10	1	3	3	7	4
24	Brazil	211	220	192	139	145	130
25	Brunei Darussalam	79	6	8	2	2	4
26	Bulgaria	24	20	12	33	11	24
27	Burkina Faso	2	1	3	2	3	2
28	Burundi	0	0	0	0	1	2
29	Cabo Verde	1	1	2	0	11	1
...
165	Suriname	15	10	21	12	5	16
166	Swaziland	4	1	1	0	10	7
167	Sweden	281	308	222	176	128	158

	Country	1980	1981	1982	1983	1984	1985
168	Switzerland	806	811	634	370	326	314
169	Syrian Arab Republic	315	419	409	269	264	385
170	Tajikistan	0	0	0	0	0	0
171	Thailand	56	53	113	65	82	66
172	The former Yugoslav Republic of Macedonia	0	0	0	0	0	0
173	Togo	5	5	2	3	6	5
174	Tonga	2	4	7	1	2	5
175	Trinidad and Tobago	958	947	972	766	606	699
176	Tunisia	58	51	55	46	51	57
177	Turkey	481	874	706	280	338	202
178	Turkmenistan	0	0	0	0	0	0
179	Tuvalu	0	1	0	0	1	0
180	Uganda	13	16	17	38	32	29
181	Ukraine	0	0	0	0	0	0
182	United Arab Emirates	0	2	2	1	2	0
183	United Kingdom of Great Britain and Northern I...	22045	24796	20620	10015	10170	9564
184	United Republic of Tanzania	635	832	621	474	473	460
185	United States of America	9378	10030	9074	7100	6661	6543
186	Uruguay	128	132	146	105	90	92
187	Uzbekistan	0	0	0	0	0	0
188	Vanuatu	0	0	0	0	0	0
189	Venezuela (Bolivarian Republic of)	103	117	174	124	142	165
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 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 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 [40]:
```

In [41]:

Out[41]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986
Country												
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2
Angola	Africa	Middle Africa	Developing regions	1	3	6	6	4	3	5
Antigua and Barbuda	Latin America and the Caribbean	Caribbean	Developing regions	0	0	0	0	42	52	51
Argentina	Latin America and the Caribbean	South America	Developing regions	368	426	626	241	237	196	213

8 rows × 38 columns

In [32]:

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 [14]:

```

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          27707
Name: Japan, dtype: object

```

In [16]:

```
495
```

In [17]:

```

1980    701
1981    756
1982    598
1983    309
1984    246
1984    246
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 ambiguity, let's convert the column names into strings: '1980' to '2013'.

```
In [18]:
```

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 [19]:
```

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 [20]:

Country	
Afghanistan	True
Albania	False
Algeria	False
American Samoa	False
Andorra	False
Angola	False
Antigua and Barbuda	False
Argentina	False
Armenia	True
Australia	False
Austria	False
Azerbaijan	True
Bahamas	False
Bahrain	True
Bangladesh	True
Barbados	False
Belarus	False
Belgium	False
Belize	False
Benin	False
Bhutan	True
Bolivia (Plurinational State of)	False
Bosnia and Herzegovina	False
Botswana	False
Brazil	False
Brunei Darussalam	True
Bulgaria	False
Burkina Faso	False
Burundi	False
Cabo Verde	False
	...
Suriname	False
Swaziland	False
Sweden	False
Switzerland	False
Syrian Arab Republic	True
Tajikistan	True
Thailand	True
The former Yugoslav Republic of Macedonia	False
Togo	False
Tonga	False
Trinidad and Tobago	False
Tunisia	False
Turkey	True
Turkmenistan	True
Tuvalu	False
Uganda	False
Ukraine	False
United Arab Emirates	True
United Kingdom of Great Britain and Northern Ireland	False
United Republic of Tanzania	False
United States of America	False
Uruguay	False
Uzbekistan	True
Vanuatu	False
Venezuela (Bolivarian Republic of)	False

Viet Nam	True
Western Sahara	False
Yemen	True
Zambia	False
Zimbabwe	False

Name: Continent, Length: 195, dtype: bool

In [21]:

Out[21]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...
Country											
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...
Armenia	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...
Azerbaijan	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...
Bahrain	Asia	Western Asia	Developing regions	0	2	1	1	1	3	0	...
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	...
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	0	...
Brunei Darussalam	Asia	South-Eastern Asia	Developing regions	79	6	8	2	2	4	12	...
Cambodia	Asia	South-Eastern Asia	Developing regions	12	19	26	33	10	7	8	...
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	...
China, Hong Kong Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...
China, Macao Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...
Cyprus	Asia	Western Asia	Developing regions	132	128	84	46	46	43	48	...
Democratic People's Republic of Korea	Asia	Eastern Asia	Developing regions	1	1	3	1	4	3	0	...
Georgia	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...
Indonesia	Asia	South-Eastern Asia	Developing regions	186	178	252	115	123	100	127	...
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	...
Iraq	Asia	Western Asia	Developing regions	262	245	260	380	428	231	265	...
Israel	Asia	Western Asia	Developing regions	1403	1711	1334	541	446	680	1212	...

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...
Country											
Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198	248	...
Jordan	Asia	Western Asia	Developing regions	177	160	155	113	102	179	181	...
Kazakhstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...
Kuwait	Asia	Western Asia	Developing regions	1	0	8	2	1	4	4	...
Kyrgyzstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...
Lao People's Democratic Republic	Asia	South-Eastern Asia	Developing regions	11	6	16	16	7	17	21	...
Lebanon	Asia	Western Asia	Developing regions	1409	1119	1159	789	1253	1683	2576	...
Malaysia	Asia	South-Eastern Asia	Developing regions	786	816	813	448	384	374	425	...
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	...
Mongolia	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...
Myanmar	Asia	South-Eastern Asia	Developing regions	80	62	46	31	41	23	18	...
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	13	...
Oman	Asia	Western Asia	Developing regions	0	0	0	8	0	0	0	...
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...
Qatar	Asia	Western Asia	Developing regions	0	0	0	0	0	0	1	...
Republic of Korea	Asia	Eastern Asia	Developing regions	1011	1456	1572	1081	847	962	1208	...
Saudi Arabia	Asia	Western Asia	Developing regions	0	0	1	4	1	2	5	...
Singapore	Asia	South-Eastern Asia	Developing regions	241	301	337	169	128	139	205	...
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	...
State of Palestine	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...
Country											
Syrian Arab Republic	Asia	Western Asia	Developing regions	315	419	409	269	264	385	493	...
Tajikistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...
Thailand	Asia	South-Eastern Asia	Developing regions	56	53	113	65	82	66	78	...
Turkey	Asia	Western Asia	Developing regions	481	874	706	280	338	202	257	...
Turkmenistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...
United Arab Emirates	Asia	Western Asia	Developing regions	0	2	2	1	2	0	5	...
Uzbekistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...
Viet Nam	Asia	South-Eastern Asia	Developing regions	1191	1829	2162	3404	7583	5907	2741	...
Yemen	Asia	Western Asia	Developing regions	1	2	1	6	0	18	7	...

49 rows × 38 columns



In [22]:

Out[22]:

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

9 rows × 38 columns



In [31]:

Out[31]:

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

9 rows × 38 columns

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

In [32]:

```
data dimensions: (195, 38)
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', 'Total'],
      dtype='object')
```

Out[32]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2013
Country												
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	34
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	10

2 rows × 38 columns

Visualizing Data using Matplotlib

Matplotlib: Standard Python Visualization Library

The primary plotting library we will explore in the course is [Matplotlib \(http://matplotlib.org/\)](http://matplotlib.org/). 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 [43]:
```

*optional: check if Matplotlib is loaded.

```
In [42]:
```

```
Matplotlib version: 2.2.2
```

*optional: apply a style to Matplotlib.

```
In [41]:
```

```
['Solarize_Light2', '_classic_test', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'seaborn', 'tableau-colorblind10']
```

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](http://pandas.pydata.org/pandas-docs/stable/api.html#plotting) (<http://pandas.pydata.org/pandas-docs/stable/api.html#plotting>)
- [Plotting with Dataframes](http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-plotting) (<http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-plotting>)

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 about 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.

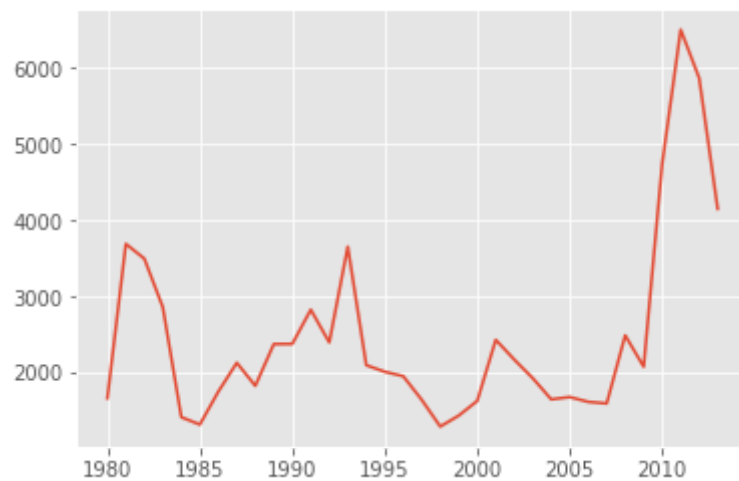
```
In [36]:
```

```
Out[36]: 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 [44]:

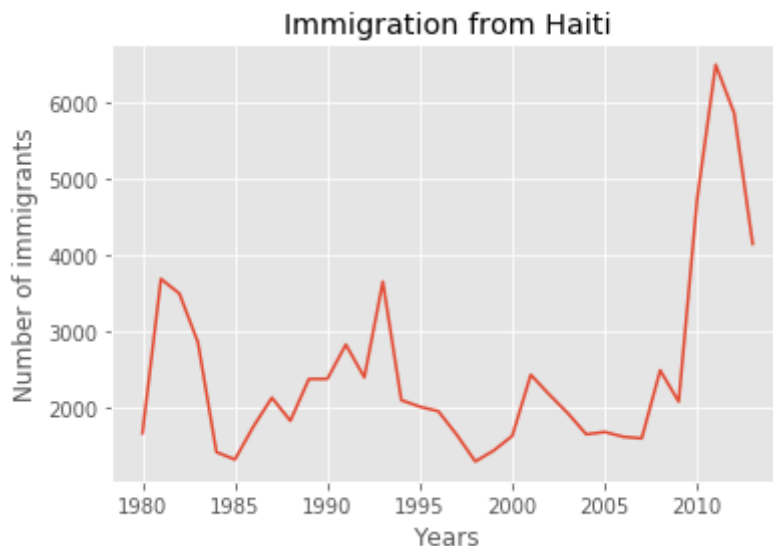
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x7f27f0feccf8>



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.

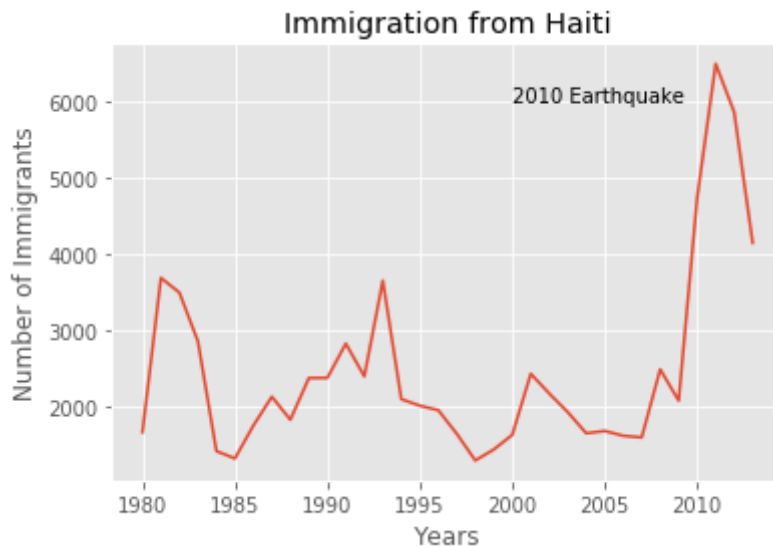
Also, let's label the x and y axis using `plt.title()`, `plt.ylabel()`, and `plt.xlabel()` as follows:

In [45]:



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 [46]:



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 [47]:

Out[47]:

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2004	2005	
Country														
China	5123	6682	3308	1863	1527	1816	1960	2643	2758	4323	...	36619	42584	3
India	8880	8670	8147	7338	5704	4211	7150	10189	11522	10343	...	28235	36210	3

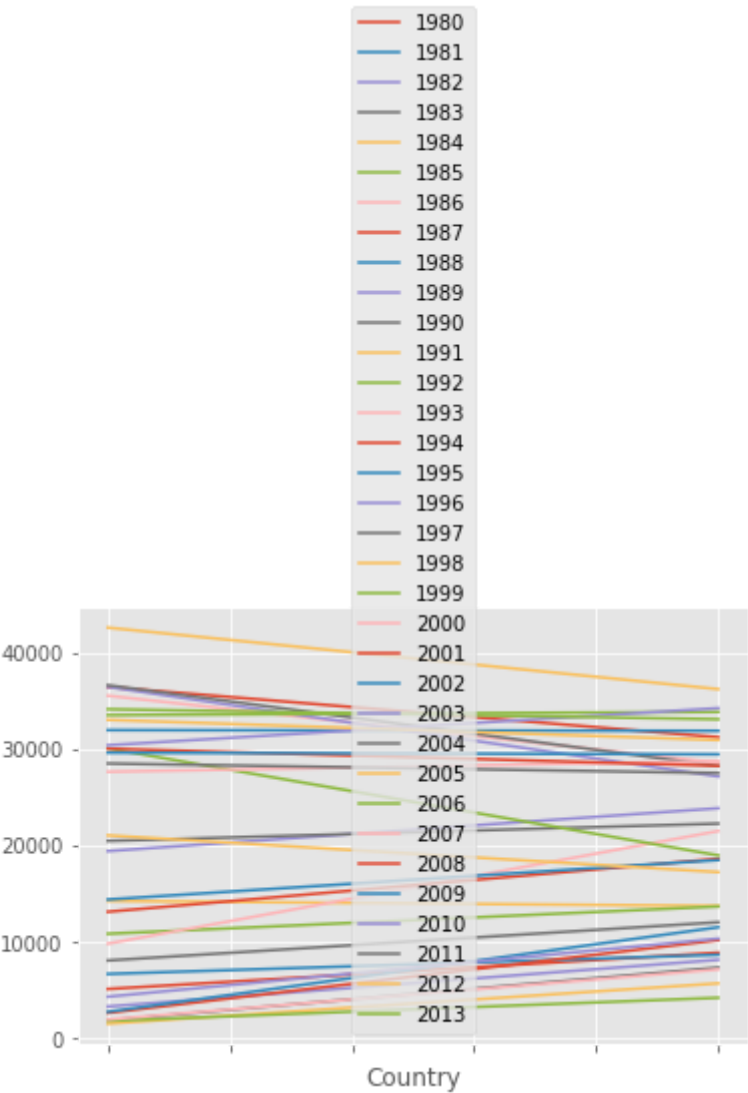
2 rows x 34 columns

Double-click **here** for the solution.

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

In [48]:

Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x7f27f0fb2320>



Double-click **here** for the 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.

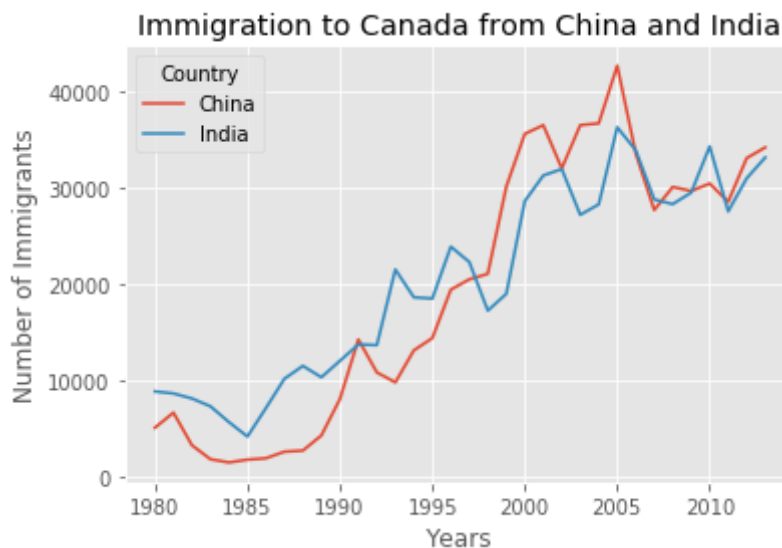
In [49]:

Out[49]:

Country	China	India
1980	5123	8880
1981	6682	8670
1982	3308	8147
1983	1863	7338
1984	1527	5704

pandas will automatically 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 [50]:



Double-click **here** for the 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.

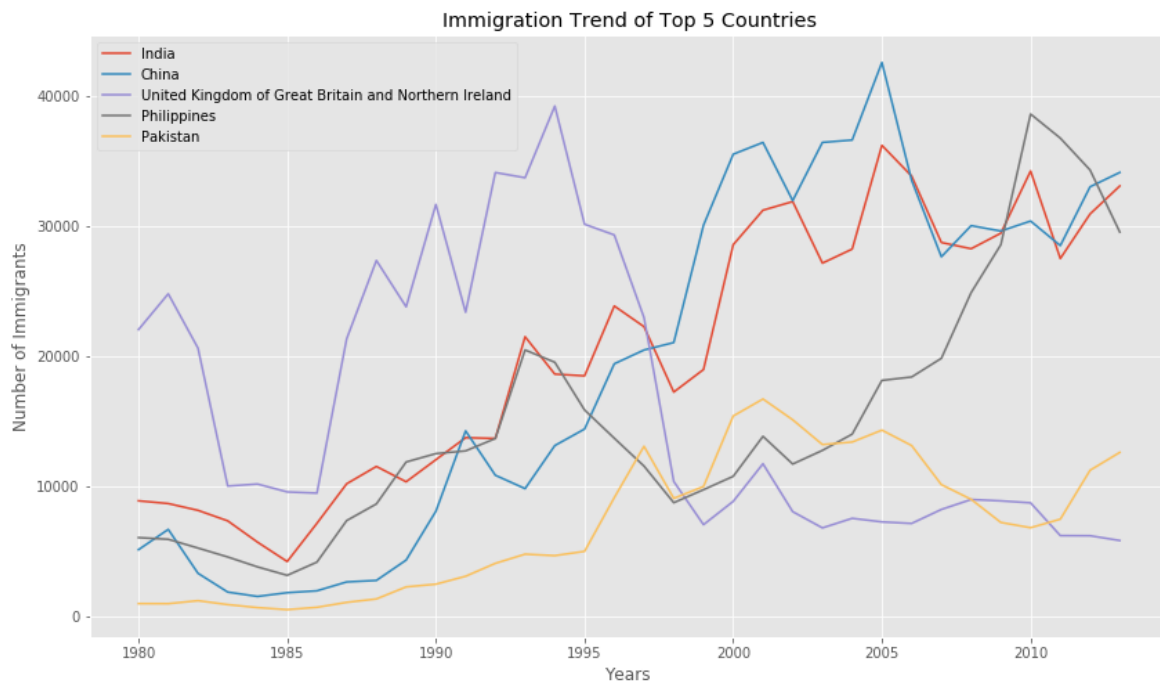
Question: Compare the trend of top 5 countries that contributed the most to immigration to Canada.

In [98]:

nd \	India	China	United Kingdom of Great Britain and Northern Ireland
1980	8880	5123	22045
1981	8670	6682	24796
1982	8147	3308	20620
1983	7338	1863	10015
1984	5704	1527	10170
1985	4211	1816	9564
1986	7150	1960	9470
1987	10189	2643	21337
1988	11522	2758	27359
1989	10343	4323	23795
1990	12041	8076	31668
1991	13734	14255	23380
1992	13673	10846	34123
1993	21496	9817	33720
1994	18620	13128	39231
1995	18489	14398	30145
1996	23859	19415	29322
1997	22268	20475	22965
1998	17241	21049	10367
1999	18974	30069	7045
2000	28572	35529	8840
2001	31223	36434	11728
2002	31889	31961	8046
2003	27155	36439	6797
2004	28235	36619	7533
2005	36210	42584	7258
2006	33848	33518	7140
2007	28742	27642	8216

2008	28261	30037	8979
2009	29456	29622	8876
2010	34235	30391	8724
2011	27509	28502	6204
2012	30933	33024	6195
2013	33087	34129	5827

	Philippines	Pakistan
1980	6051	978
1981	5921	972
1982	5249	1201
1983	4562	900
1984	3801	668
1985	3150	514
1986	4166	691
1987	7360	1072
1988	8639	1334
1989	11865	2261
1990	12509	2470
1991	12718	3079
1992	13670	4071
1993	20479	4777
1994	19532	4666
1995	15864	4994
1996	13692	9125
1997	11549	13073
1998	8735	9068
1999	9734	9979
2000	10763	15400
2001	13836	16708
2002	11707	15110
2003	12758	13205
2004	14004	13399
2005	18139	14314
2006	18400	13127
2007	19837	10124
2008	24887	8994
2009	28573	7217
2010	38617	6811
2011	36765	7468
2012	34315	11227
2013	29544	12603



Double-click **here** for the solution.

In [99]:

In [100]:

In [101]:

In [102]:

In [103]:

In [104]:

In [105]:

In [106]:

In [107]:

In [108]:

In [109]:

In [110]:

In [111]:

```
In [112]:
```

```
In [113]:
```

```
In [114]:
```

```
In [115]:
```

```
In [116]:
```

```
In [117]:
```

In [124]:

India	True
China	True
United Kingdom of Great Britain and Northern Ireland	False
Philippines	True
Pakistan	True
United States of America	False
Iran (Islamic Republic of)	True
Sri Lanka	True
Republic of Korea	True
Poland	False
Lebanon	True
France	False
Jamaica	True
Viet Nam	True
Romania	False
Haiti	True
Guyana	True
Portugal	False
Egypt	True
Morocco	True
Colombia	True
Iraq	True
Algeria	True
Israel	True
Bangladesh	True
Germany	False
Russian Federation	False
Mexico	True
Afghanistan	True
El Salvador	True
...	
Oman	True
Luxembourg	False
Cabo Verde	True
Comoros	True
Swaziland	True
Mozambique	True
Qatar	True
Tonga	True
Lesotho	True
Montenegro	False
Guinea-Bissau	True
Samoa	True
Papua New Guinea	True
Equatorial Guinea	True
Liechtenstein	False
Maldives	True
Monaco	False
Canada	False
Nauru	True
Andorra	False
Kiribati	True
Vanuatu	True
Sao Tome and Principe	True
Tuvalu	True
American Samoa	True
San Marino	False

New Caledonia	True
Marshall Islands	True
Western Sahara	True
Palau	True
Name: DevName, Length: 195, dtype: bool	

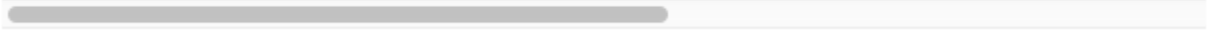
Out[124]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	..
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	..
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	..
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	..
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	..
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	..
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	..
Republic of Korea	Asia	Eastern Asia	Developing regions	1011	1456	1572	1081	847	962	1208	..
Lebanon	Asia	Western Asia	Developing regions	1409	1119	1159	789	1253	1683	2576	..
Jamaica	Latin America and the Caribbean	Caribbean	Developing regions	3198	2634	2661	2455	2508	2938	4649	..
Viet Nam	Asia	South-Eastern Asia	Developing regions	1191	1829	2162	3404	7583	5907	2741	..
Haiti	Latin America and the Caribbean	Caribbean	Developing regions	1666	3692	3498	2860	1418	1321	1753	..
Guyana	Latin America and the Caribbean	South America	Developing regions	2334	2943	3575	2650	1932	2299	3942	..
Egypt	Africa	Northern Africa	Developing regions	612	660	755	455	447	348	514	..
Morocco	Africa	Northern Africa	Developing regions	325	471	447	335	248	328	388	..
Colombia	Latin America and the Caribbean	South America	Developing regions	266	326	360	244	235	214	257	..
Iraq	Asia	Western Asia	Developing regions	262	245	260	380	428	231	265	..
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	..
Israel	Asia	Western Asia	Developing regions	1403	1711	1334	541	446	680	1212	..
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	..

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	..
Mexico	Latin America and the Caribbean	Central America	Developing regions	409	394	491	490	509	425	667	..
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	..
El Salvador	Latin America and the Caribbean	Central America	Developing regions	110	295	882	2587	2666	2769	3106	..
Trinidad and Tobago	Latin America and the Caribbean	Caribbean	Developing regions	958	947	972	766	606	699	955	..
Somalia	Africa	Eastern Africa	Developing regions	7	10	7	12	14	10	37	..
South Africa	Africa	Southern Africa	Developing regions	1026	1118	781	379	271	310	718	..
Nigeria	Africa	Western Africa	Developing regions	81	60	58	58	78	78	114	..
Jordan	Asia	Western Asia	Developing regions	177	160	155	113	102	179	181	..
Ethiopia	Africa	Eastern Africa	Developing regions	98	59	54	98	154	157	302	..
Peru	Latin America and the Caribbean	South America	Developing regions	317	456	401	241	306	328	628	..
Turkey	Asia	Western Asia	Developing regions	481	874	706	280	338	202	257	..
...
Bahrain	Asia	Western Asia	Developing regions	0	2	1	1	1	3	0	..
Botswana	Africa	Southern Africa	Developing regions	10	1	3	3	7	4	2	..
Democratic People's Republic of Korea	Asia	Eastern Asia	Developing regions	1	1	3	1	4	3	0	..
Namibia	Africa	Southern Africa	Developing regions	0	5	5	3	2	1	1	..
Turkmenistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	..
Malawi	Africa	Eastern Africa	Developing regions	5	4	6	3	2	0	4	..
China, Macao Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	..

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	..
Oman	Asia	Western Asia	Developing regions	0	0	0	8	0	0	0	..
Cabo Verde	Africa	Western Africa	Developing regions	1	1	2	0	11	1	9	..
Comoros	Africa	Eastern Africa	Developing regions	0	2	2	0	0	2	1	..
Swaziland	Africa	Southern Africa	Developing regions	4	1	1	0	10	7	1	..
Mozambique	Africa	Eastern Africa	Developing regions	0	0	7	2	3	1	3	..
Qatar	Asia	Western Asia	Developing regions	0	0	0	0	0	0	1	..
Tonga	Oceania	Polynesia	Developing regions	2	4	7	1	2	5	7	..
Lesotho	Africa	Southern Africa	Developing regions	1	1	1	2	7	5	3	..
Guinea-Bissau	Africa	Western Africa	Developing regions	0	0	0	0	1	0	0	..
Samoa	Oceania	Polynesia	Developing regions	3	7	4	1	3	0	3	..
Papua New Guinea	Oceania	Melanesia	Developing regions	0	2	2	4	2	2	1	..
Equatorial Guinea	Africa	Middle Africa	Developing regions	0	0	0	0	0	0	1	..
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	..
Nauru	Oceania	Micronesia	Developing regions	1	0	0	0	0	0	0	..
Kiribati	Oceania	Micronesia	Developing regions	0	0	0	1	0	0	0	..
Vanuatu	Oceania	Melanesia	Developing regions	0	0	0	0	0	0	0	..
Sao Tome and Principe	Africa	Middle Africa	Developing regions	0	0	0	0	0	0	0	..
Tuvalu	Oceania	Polynesia	Developing regions	0	1	0	0	1	0	0	..
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0	..
New Caledonia	Oceania	Melanesia	Developing regions	0	0	0	0	0	0	0	..
Marshall Islands	Oceania	Micronesia	Developing regions	0	0	0	0	0	0	0	..
Western Sahara	Africa	Northern Africa	Developing regions	0	0	0	0	0	0	0	..
Palau	Oceania	Micronesia	Developing regions	0	0	0	0	0	0	0	..

147 rows × 38 columns



In [125]:

In [126]:

In [127]:

In [128]:

In [129]:

In [130]:

In [131]:

In [132]:

In [133]:

In [134]:

In [135]:

In [136]:

In [137]:

In [138]:

In [139]:

In [140]:

In [141]:

In [142]:

In [143]:

In [144]:

In [145]:

In [146]:

In [147]:

In [148]:

In [149]:

In [150]:

In [151]:

In [152]:

In [153]:

In [154]:

In [155]:

In [156]:

In [157]:

In [158]:

In [159]:

In [160]:

In [161]:

In [162]:

In [163]:

In [164]:

In [165]:

In [166]:

In [167]:

In [168]:

In [169]:

In [170]:

In [171]:

In [172]:

In [173]:

In [174]:

In [175]:

In [176]:

In [177]:

In [178]:

In [179]:

In [180]:

In [181]:

In [182]:

In [183]:

In [184]:

In [185]:

In [186]:

In [187]:

In [188]:

In [189]:

In [190]:

In [191]:

In [192]:

In [193]:

In [194]:

In [195]:

In [196]:

In [197]:

In [198]:

In [199]:

In [200]:

In [201]:

In [202]:

In [203]:

In [204]:

In [205]:

In [206]:

In [207]:

In [208]:

In [209]:

In [210]:

In [211]:

In [212]:

In [213]:

In [214]:

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!

This notebook was originally created by [Jay Rajasekharan](https://www.linkedin.com/in/jayrajasekharan) with contributions from [Ehsan M. Kermani](https://www.linkedin.com/in/ehsanmkermani), and [Slobodan Markovic](https://www.linkedin.com/in/slobodan-markovic).

This notebook was recently revised by [Alex Akison](https://www.linkedin.com/in/aklson/). I hope you found this lab session interesting. Feel free to contact me if you have any questions!

This notebook is part of a course on **Coursera** called *Data Visualization with Python*. If you accessed this notebook outside the course, you can take this course online by clicking [here](http://cocl.us/DV0101EN_Coursera_Week1_LAB1) (http://cocl.us/DV0101EN_Coursera_Week1_LAB1).

Copyright © 2018 [Cognitive Class](https://cognitiveclass.ai/?utm_source=bducopyrightlink&utm_medium=dswb&utm_campaign=bdu). This notebook and its source code are released under the terms of the [MIT License](https://bigdatauniversity.com/mit-license/).

In [215]:

Out[215]:

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

5 rows × 38 columns



In [217]:

In [219]:

Out[219]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986
United Kingdom of Great Britain and Northern Ireland	Europe	Northern Europe	Developed regions	22045	24796	20620	10015	10170	9564	9470
United States of America	Northern America	Northern America	Developed regions	9378	10030	9074	7100	6661	6543	7074
Poland	Europe	Eastern Europe	Developed regions	863	2930	5881	4546	3588	2819	4808
France	Europe	Western Europe	Developed regions	1729	2027	2219	1490	1169	1177	1298
Romania	Europe	Eastern Europe	Developed regions	375	438	583	543	524	604	656
Portugal	Europe	Southern Europe	Developed regions	4473	3486	2432	1433	1398	1451	2617
Germany	Europe	Western Europe	Developed regions	1626	1977	3062	2376	1610	1441	1233
Russian Federation	Europe	Eastern Europe	Developed regions	0	0	0	0	0	0	0
Ukraine	Europe	Eastern Europe	Developed regions	0	0	0	0	0	0	0
Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198	248
Netherlands	Europe	Western Europe	Developed regions	1889	1858	1852	716	560	510	539
Australia	Oceania	Australia and New Zealand	Developed regions	702	639	484	317	317	319	356
Bulgaria	Europe	Eastern Europe	Developed regions	24	20	12	33	11	24	33
Italy	Europe	Southern Europe	Developed regions	1820	2057	1480	820	858	667	731
Bosnia and Herzegovina	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0
Ireland	Europe	Northern Europe	Developed regions	781	895	707	298	327	287	481
Hungary	Europe	Eastern Europe	Developed regions	205	310	397	337	310	522	647
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1
Switzerland	Europe	Western Europe	Developed regions	806	811	634	370	326	314	294
Republic of Moldova	Europe	Eastern Europe	Developed regions	0	0	0	0	0	0	0
Greece	Europe	Southern Europe	Developed regions	1065	953	897	633	580	584	547

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986
Belgium	Europe	Western Europe	Developed regions	511	540	519	297	183	181	197
New Zealand	Oceania	Australia and New Zealand	Developed regions	602	480	364	140	164	148	176
Croatia	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0
Belarus	Europe	Eastern Europe	Developed regions	0	0	0	0	0	0	0
Slovakia	Europe	Eastern Europe	Developed regions	0	0	0	0	0	0	0
Sweden	Europe	Northern Europe	Developed regions	281	308	222	176	128	158	187
Spain	Europe	Southern Europe	Developed regions	211	299	260	133	137	98	121
Austria	Europe	Western Europe	Developed regions	234	238	201	117	127	165	196
The former Yugoslav Republic of Macedonia	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0
Czech Republic	Europe	Eastern Europe	Developed regions	0	0	0	0	0	0	0
Denmark	Europe	Northern Europe	Developed regions	272	293	299	106	93	73	93
Finland	Europe	Northern Europe	Developed regions	208	205	170	70	83	69	68
Lithuania	Europe	Northern Europe	Developed regions	1	1	0	0	0	1	0
Norway	Europe	Northern Europe	Developed regions	116	77	106	51	31	54	56
Latvia	Europe	Northern Europe	Developed regions	0	0	0	0	0	0	1
Serbia	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0
Malta	Europe	Southern Europe	Developed regions	191	242	153	64	60	68	76
Estonia	Europe	Northern Europe	Developed regions	0	0	0	0	0	0	0
Iceland	Europe	Northern Europe	Developed regions	17	33	10	9	13	6	11
Slovenia	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0
Luxembourg	Europe	Western Europe	Developed regions	14	4	2	5	1	3	6
Montenegro	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986
Liechtenstein	Europe	Western Europe	Developed regions	1	4	2	0	0	3	0
Monaco	Europe	Western Europe	Developed regions	0	0	0	0	0	1	0
Canada	Northern America	Northern America	Developed regions	0	0	0	0	0	0	0
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2
San Marino	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1

48 rows × 38 columns

In [222]:

```
-----
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-222-aad18e61aaf3> in <module>()
----> 1 test2.sort_values(by='Total', ascending=False, axis=0, inplace
=True)
      2 df_top10 = test2.head(5)
      3 df_top7 = df_top7[years].transpose()
      4 print(df_top7)
      5 print('\n')

TypeError: sort_values() got an unexpected keyword argument 'by'
```

In []: