(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 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 for Data Science** (http://cocl.us/PY0101EN_DV0101EN_LAB1_Coursera) and **Data Analysis with Python** (http://cocl.us/DA0101EN_DV0101EN_LAB1_Coursera), which are 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), which is also part of this specialization.

Table of Contents

Exploring Datasets with *pandas*

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

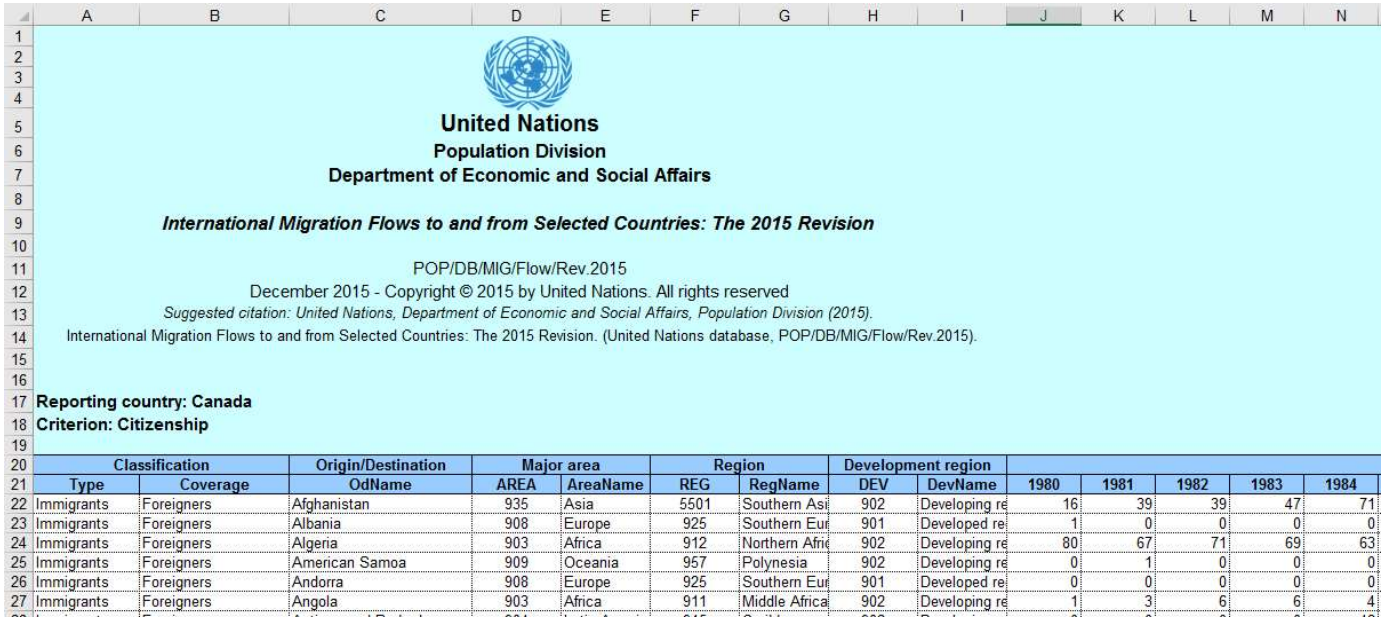
The course heavily relies on *pandas* for data wrangling, analysis, and visualization. We encourage you to spend some time and familizare yourself with the *pandas* API Reference: http://pandas.pydata.org/pandas-docs/stable/api.html (http://pandas.pydata.org/pandas-docs/stable/api.html).

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.



United Nations
Population Division
Department of Economic and Social Affairs

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

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

Reporting country: Canada
Criterion: Citizenship

Classification		Origin/Destination	Major area		Region		Development region						
Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	1981	1982	1983	1984
Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asi	902	Developing re	16	39	39	47	71
Immigrants	Foreigners	Albania	908	Europe	925	Southern Eur	901	Developed re	1	0	0	0	0
Immigrants	Foreigners	Algeria	903	Africa	912	Northern Afric	902	Developing re	80	67	71	69	63
Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing re	0	1	0	0	0
Immigrants	Foreigners	Andorra	908	Europe	925	Southern Eur	901	Developed re	0	0	0	0	0
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 [1]:

In [9]:

```
Requirement already satisfied: xlrd in /home/jupyterlab/conda/envs/python/li
b/python3.6/site-packages (1.2.0)
Note: you may need to restart the kernel to use updated packages.
```

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 [10]:

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

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

In [7]:

Out[7]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	19
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	
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 veiw the bottom 5 rows of the dataset using the `tail()` function.

In [11]:

Out[11]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1
190	Immigrants	Foreigners	Viet Nam	935	Asia	920	South-Eastern Asia	902	Developing regions	1
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 [12]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 43 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Type      195 non-null    object
 1   Coverage  195 non-null    object
 2   OdName    195 non-null    object
 3   AREA      195 non-null    int64 
 4   AreaName  195 non-null    object
 5   REG       195 non-null    int64 
 6   RegName   195 non-null    object
 7   DEV       195 non-null    int64 
 8   DevName   195 non-null    object
 9   1980      195 non-null    int64 
 10  1981      195 non-null    int64 
 11  1982      195 non-null    int64 
 12  1983      195 non-null    int64 
 13  1984      195 non-null    int64 
 14  1985      195 non-null    int64 
 15  1986      195 non-null    int64 
 16  1987      195 non-null    int64 
 17  1988      195 non-null    int64 
 18  1989      195 non-null    int64 
 19  1990      195 non-null    int64 
 20  1991      195 non-null    int64 
 21  1992      195 non-null    int64 
 22  1993      195 non-null    int64 
 23  1994      195 non-null    int64 
 24  1995      195 non-null    int64 
 25  1996      195 non-null    int64 
 26  1997      195 non-null    int64 
 27  1998      195 non-null    int64 
 28  1999      195 non-null    int64 
 29  2000      195 non-null    int64 
 30  2001      195 non-null    int64 
 31  2002      195 non-null    int64 
 32  2003      195 non-null    int64 
 33  2004      195 non-null    int64 
 34  2005      195 non-null    int64 
 35  2006      195 non-null    int64 
 36  2007      195 non-null    int64 
 37  2008      195 non-null    int64 
 38  2009      195 non-null    int64 
 39  2010      195 non-null    int64 
 40  2011      195 non-null    int64 
 41  2012      195 non-null    int64 
 42  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 [13]:

Out[13]:

```
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 indicies we use the `.index` parameter.

In [14]:

Out[14]:

```
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 [15]:

```
<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 [16]:

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

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

In [17]:

Out[17]:

```
(195, 43)
```

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 [18]:

Out[18]:

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	...	200
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	297
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	145

2 rows × 38 columns

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

In [19]:

Out[19]:

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

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

In [20]:

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

In [21]:

Out[21]:

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

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

In [22]:

Out[22]:

	1980	1981	1982	1983	1984	1985	
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	1
mean	508.394872	566.989744	534.723077	387.435897	376.497436	358.861538	4
std	1949.588546	2152.643752	1866.997511	1204.333597	1198.246371	1079.309600	12
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	13.000000	10.000000	11.000000	12.000000	13.000000	17.000000	
75%	251.500000	295.500000	275.000000	173.000000	181.000000	197.000000	2
max	22045.000000	24796.000000	20620.000000	10015.000000	10170.000000	9564.000000	94

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 [23]:

Out[23]:

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

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

In [24]:

Out[24]:

	Country	1980	1981	1982	1983	1984	1985
0	Afghanistan	16	39	39	47	71	340
1	Albania	1	0	0	0	0	0
2	Algeria	80	67	71	69	63	44
3	American Samoa	0	1	0	0	0	0
4	Andorra	0	0	0	0	0	0
...	...	...	...	...	...	...	...
190	Viet Nam	1191	1829	2162	3404	7583	5907
191	Western Sahara	0	0	0	0	0	0
192	Yemen	1	2	1	6	0	18
193	Zambia	11	17	11	7	16	9
194	Zimbabwe	72	114	102	44	32	29

195 rows × 7 columns

Select Row

There are main 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 [33]:

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-33-9eb621aa398f> in <module>
----> 1 df_can.set_index('Country', inplace=True)
      2 # tip: The opposite of set is reset. So to reset the index, we can u
se df_can.reset_index()

~/conda/envs/python/lib/python3.6/site-packages/pandas/core/frame.py in set_
index(self, keys, drop, append, inplace, verify_integrity)
   4298
   4299         if missing:
-> 4300             raise KeyError(f"None of {missing} are in the columns")
   4301
   4302         if inplace:

KeyError: "None of ['Country'] are in the columns"
```

In [34]:

Out[34]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3

3 rows × 38 columns

In [27]:

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 [28]:

```
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
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
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 [29]:

```
982
982
```

In [30]:

```
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 [35]:

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 [32]:

Out[32]:

```
['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 [ ]:

In [ ]:

In [ ]:

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

In [36]:

```
data dimensions: (195, 38)
Index(['Continent', 'Region', 'DevName', '1980', '1981', '1982', '1983',
       '1984', '1985', '1986', '1987', '1988', '1989', '1990', '1991', '199
2',
       '1993', '1994', '1995', '1996', '1997', '1998', '1999', '2000', '200
1',
       '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '201
0',
       '2011', '2012', '2013', 'Total'],
      dtype='object')
```

Out[36]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1

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/). 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 [37]:

*optional: check if Matplotlib is loaded.

In [38]:

```
Matplotlib version:  3.1.0
```

*optional: apply a style to Matplotlib.

In [39]:

```
['fivethirtyeight', 'seaborn-ticks', 'seaborn-pastel', 'seaborn-notebook',
'dark_background', 'seaborn-white', 'seaborn-talk', '_classic_test', 'bmh',
'seaborn-paper', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-colorblin
d', 'seaborn-whitegrid', 'tableau-colorblind10', 'seaborn-darkgrid', 'graysc
ale', 'seaborn-bright', 'classic', 'seaborn-poster', 'ggplot', 'fast', 'seab
orn-muted', 'Solarize_Light2', 'seaborn', 'seaborn-deep']
```

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)
- Plotting with Dataframes (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 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 [41]:

Out[41]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3
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	...	3
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	...	

5 rows × 38 columns

In [42]:

Out[42]:

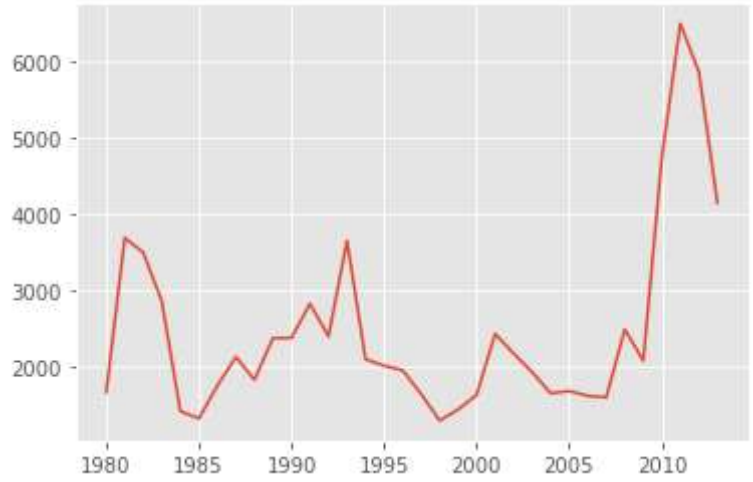
```
1980    1666
1981    3692
1982    3498
1983    2860
1984    1418
Name: Haiti, dtype: object
```

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

In [43]:

Out[43]:

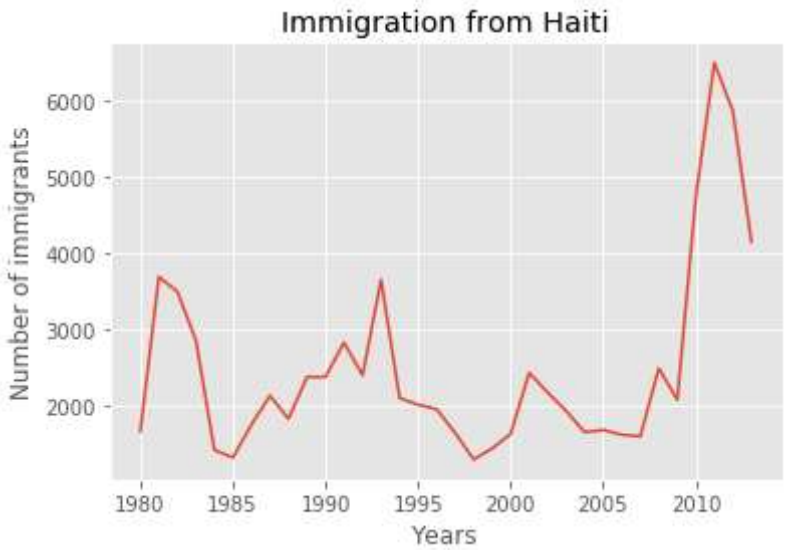
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc628aa3eb8>
```



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.

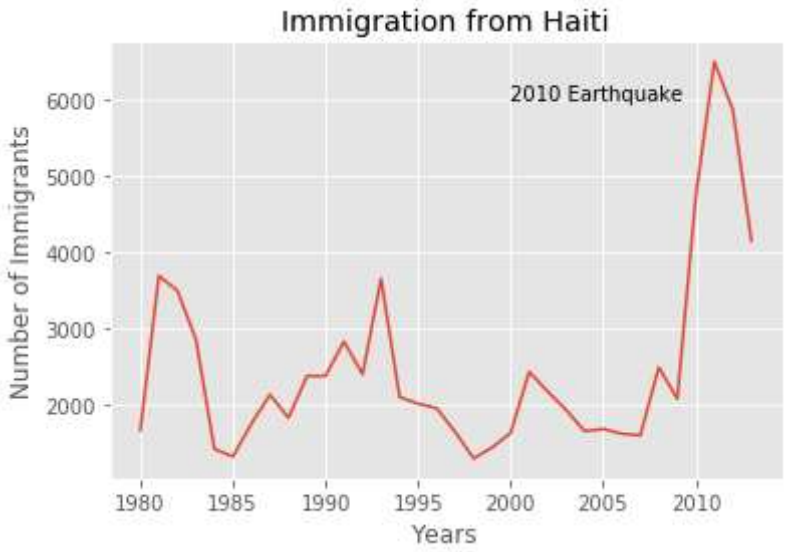
Also, let's label the x and y axis using `plt.title()`, `plt.ylabel()`, and `plt.xlabel()` as follows:

In [44]:



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 [45]:



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 = 60
00.


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 [55]:

Out[55]:

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2004	2005	2(
India	8880	8670	8147	7338	5704	4211	7150	10189	11522	10343	...	28235	36210	33
China	5123	6682	3308	1863	1527	1816	1960	2643	2758	4323	...	36619	42584	33

2 rows × 34 columns

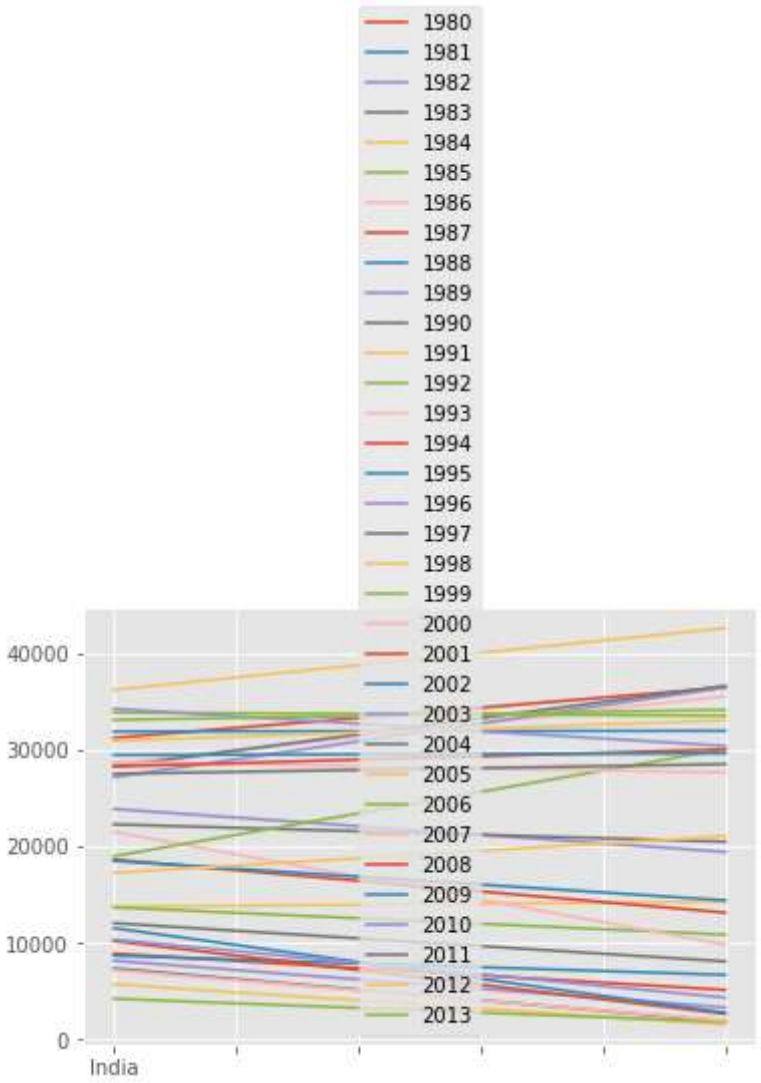
Double-click **here** for the solution.

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

In [53]:

Out[53]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc628506ac8>
```



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.

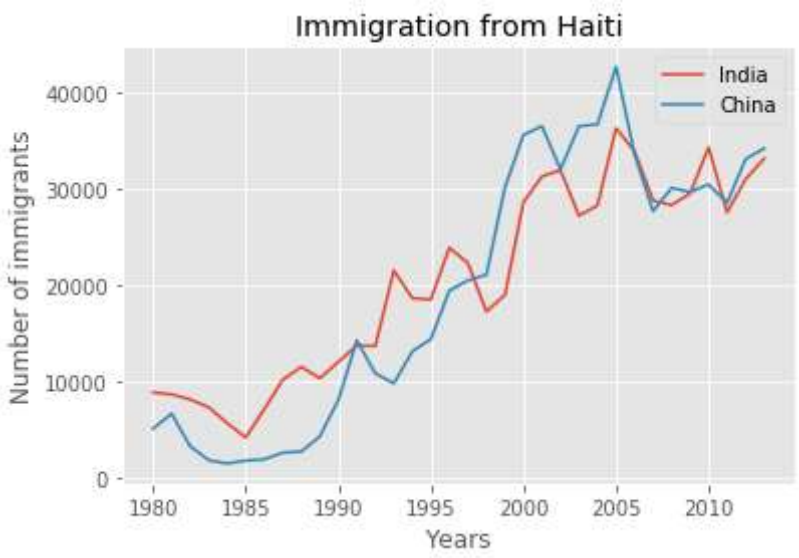
In [56]:

Out[56]:

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

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 [60]:



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.

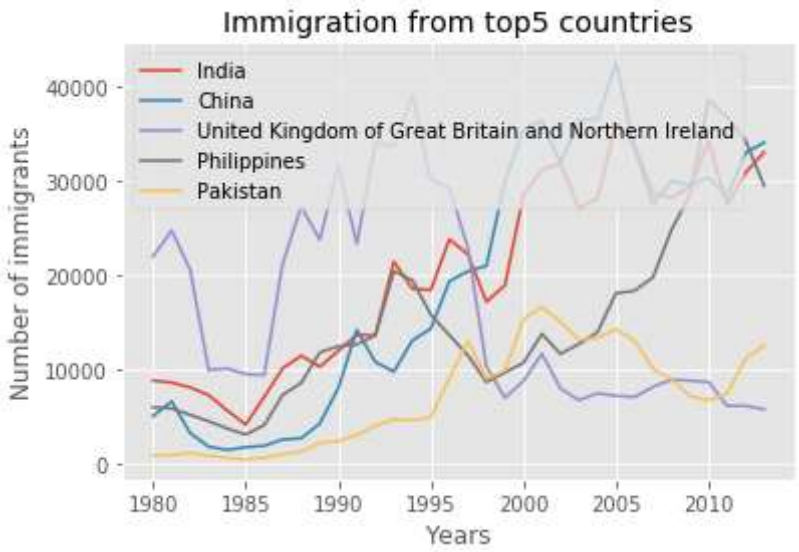
```
print(type(haiti))
print(haiti.head(5))
```

```
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 [62]:



Double-click **here** for the 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!

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 Aklson (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).
