DV0101EN-1-1-1-Introduction-to-Matplotlib-and-Line-Plots

January 26, 2021

1 Data Visualization

Estimated time needed: **30** minutes

1.1 Objectives

After completing this lab you will be able to:

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

1.2 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**.

1.3 Table of Contents

1. Section **??** 1.1 Section **??** 1.2 Section **??** 1.3 Section **??**
2. Section **??** 2.1 Section **??**
3. Section **??**

2 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.

2.1 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.

The Canada Immigration dataset can be fetched from here.

2.2 *pandas* Basics

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

[ ]:
```
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.

[2]:
```
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
df_can = pd.read_excel('https://cf-courses-data.s3.us.cloud-object-storage.
 ↪appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/
 ↪Data%20Files/Canada.xlsx',
                        sheet_name='Canada by Citizenship',
                       skiprows=range(20),
                       skipfooter=2)

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

Data read into a pandas dataframe!

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

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

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

[3]:
```
df_can.tail()
```

[3]:

	Type	Coverage	OdName	AREA	AreaName	REG	\
190	Immigrants	Foreigners	Viet Nam	935	Asia	920	
191	Immigrants	Foreigners	Western Sahara	903	Africa	912	
192	Immigrants	Foreigners	Yemen	935	Asia	922	
193	Immigrants	Foreigners	Zambia	903	Africa	910	
194	Immigrants	Foreigners	Zimbabwe	903	Africa	910	

	RegName	DEV	DevName	1980	…	2004	2005	2006	\
190	South-Eastern Asia	902	Developing regions	1191	…	1816	1852	3153	
191	Northern Africa	902	Developing regions	0	…	0	0	1	
192	Western Asia	902	Developing regions	1	…	124	161	140	
193	Eastern Africa	902	Developing regions	11	…	56	91	77	
194	Eastern Africa	902	Developing regions	72	…	1450	615	454	

	2007	2008	2009	2010	2011	2012	2013
190	2574	1784	2171	1942	1723	1731	2112
191	0	0	0	0	0	0	0
192	122	133	128	211	160	174	217
193	71	64	60	102	69	46	59
194	663	611	508	494	434	437	407

[5 rows x 43 columns]

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

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

[4]:
```
df_can.info(verbose=False)
```

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

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

[5]:
```
df_can.columns.values
```

[5]:
```
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.

[6]:
```
df_can.index.values
```

[6]:
```
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.

[1]:
```
print(type(df_can.columns))
print(type(df_can.index))
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-1-ff96affd87c1> in <module>
----> 1 print(type(df_can.columns))
      2 print(type(df_can.index))


NameError: name 'df_can' is not defined
```

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

[8]:
```
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.

[9]:
```
# size of dataframe (rows, columns)
df_can.shape
```

[9]:
```
(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:

[18]:
```
# in pandas axis=0 represents rows (default) and axis=1 represents columns.
df_can.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=True)
df_can.head(2)
```

[18]:

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	…	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	…	3436	3009	2652	2111	1746	1758	2203	2635	2004	65977
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	…	1223	856	702	560	716	561	539	620	603	18433

[2 rows x 39 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:

[19]:
```
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'RegName':
 ↪'Region'}, inplace=True)
df_can.columns
```

[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,     'Total'],
      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:

[16]:
```
df_can['Total'] = df_can.sum(axis=1)
```

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

[15]:
```
df_can.isnull().sum()
```

[15]:
```
Type        0
Coverage    0
OdName      0
AREA        0
AreaName    0
REG         0
RegName     0
DEV         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
dtype: int64
```

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

[14]:
```
df_can.describe()
```

[14]:
```
              AREA          REG          DEV          1980          1981  \
count   195.000000   195.000000   195.000000    195.000000    195.000000
mean    912.764103  1249.015385   901.753846    508.394872    566.989744
std      13.082835  1185.526885     0.431878   1949.588546   2152.643752
min     903.000000   905.000000   901.000000      0.000000      0.000000
25%     903.000000   914.000000   902.000000      0.000000      0.000000
50%     908.000000   922.000000   902.000000     13.000000     10.000000
75%     922.000000   925.500000   902.000000    251.500000    295.500000
max     935.000000  5501.000000   902.000000  22045.000000  24796.000000

              1982          1983          1984         1985          1986  \
count   195.000000    195.000000    195.000000   195.000000    195.000000
mean    534.723077    387.435897    376.497436   358.861538    441.271795
std    1866.997511   1204.333597   1198.246371  1079.309600   1225.576630
min       0.000000      0.000000      0.000000     0.000000      0.000000
25%       0.000000      0.000000      0.000000     0.000000      0.500000
50%      11.000000     12.000000     13.000000    17.000000     18.000000
75%     275.000000    173.000000    181.000000   197.000000    254.000000
max   20620.000000  10015.000000  10170.000000  9564.000000  9470.000000

       …          2004          2005          2006          2007  \
count  …    195.000000    195.000000    195.000000    195.000000
mean   …   1190.169231   1320.292308   1266.958974   1191.820513
std    …   3710.505369   4425.957828   3926.717747   3443.542409
min    …      0.000000      0.000000      0.000000      0.000000
25%    …     19.000000     28.500000     25.000000     31.000000
50%    …    191.000000    210.000000    218.000000    198.000000
```

```
75%    …    756.500000    832.000000    842.000000    899.000000
max    …  36619.000000  42584.000000  33848.000000  28742.000000

              2008          2009          2010          2011          2012  \
count   195.000000    195.000000    195.000000    195.000000    195.000000
mean   1246.394872   1275.733333   1420.287179   1262.533333   1313.958974
std    3694.573544   3829.630424   4462.946328   4030.084313   4247.555161
min       0.000000      0.000000      0.000000      0.000000      0.000000
25%      31.000000     36.000000     40.500000     37.500000     42.500000
50%     205.000000    214.000000    211.000000    179.000000    233.000000
75%     934.500000    888.000000    932.000000    772.000000    783.000000
max   30037.000000  29622.000000  38617.000000  36765.000000  34315.000000

              2013
count   195.000000
mean   1320.702564
std    4237.951988
min       0.000000
25%      45.000000
50%     213.000000
75%     796.000000
max   34129.000000

[8 rows x 37 columns]
```

2.3 *pandas* Intermediate: Indexing and Selection (slicing)

2.3.1 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').

[21]:
```
df_can.Country  # returns a series
```

[21]:
```
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.

[22]:
```
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.
```

[22]:

	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 x 7 columns]

2.3.2 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.

[23]:
```
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()
```

[24]:
```
df_can.head(3)
```

[24]:
```
           Continent           Region            DevName  1980  1981  1982  \
Country
Afghanistan      Asia    Southern Asia  Developing regions    16    39    39
Albania        Europe  Southern Europe   Developed regions     1     0     0
Algeria        Africa  Northern Africa  Developing regions    80    67    71

             1983  1984  1985  1986  …  2005  2006  2007  2008  2009  2010  \
Country                              …
Afghanistan    47    71   340   496  …  3436  3009  2652  2111  1746  1758
Albania         0     0     0     1  …  1223   856   702   560   716   561
Algeria        69    63    44    69  …  3626  4807  3623  4005  5393  4752

             2011  2012  2013  Total
Country
Afghanistan  2203  2635  2004  65977
Albania       539   620   603  18433
Algeria      4325  3774  4331  72156

[3 rows x 38 columns]
```

[25]:
```
# optional: to remove the name of the index
df_can.index.name = None
df_can
```

[25]:
```
               Continent             Region            DevName  1980  1981  \
Afghanistan         Asia       Southern Asia  Developing regions    16    39
Albania           Europe     Southern Europe   Developed regions     1     0
Algeria           Africa     Northern Africa  Developing regions    80    67
American Samoa   Oceania           Polynesia  Developing regions     0     1
Andorra           Europe     Southern Europe   Developed regions     0     0
…                    …                   …                  …   …   …
Viet Nam            Asia  South-Eastern Asia  Developing regions  1191  1829
Western Sahara    Africa     Northern Africa  Developing regions     0     0
Yemen               Asia        Western Asia  Developing regions     1     2
Zambia            Africa      Eastern Africa  Developing regions    11    17
Zimbabwe          Africa      Eastern Africa  Developing regions    72   114

                1982  1983  1984  1985  1986  …  2005  2006  2007  2008  \
Afghanistan       39    47    71   340   496  …  3436  3009  2652  2111
```

```
Albania             0     0     0     0     1  …  1223   856   702   560
Algeria            71    69    63    44    69  …  3626  4807  3623  4005
American Samoa      0     0     0     0     0  …     0     1     0     0
Andorra             0     0     0     0     2  …     0     1     1     0
…                 …   …     …     …   …     …  …   …     …     …     …
Viet Nam         2162  3404  7583  5907  2741  …  1852  3153  2574  1784
Western Sahara      0     0     0     0     0  …     0     1     0     0
Yemen               1     6     0    18     7  …   161   140   122   133
Zambia             11     7    16     9    15  …    91    77    71    64
Zimbabwe          102    44    32    29    43  …   615   454   663   611

                 2009  2010  2011  2012  2013  Total
Afghanistan      1746  1758  2203  2635  2004  65977
Albania           716   561   539   620   603  18433
Algeria          5393  4752  4325  3774  4331  72156
American Samoa      0     0     0     0     0   2774
Andorra             0     0     0     1     1   2749
…                 …   …     …     …   …     …
Viet Nam         2171  1942  1723  1731  2112  99903
Western Sahara      0     0     0     0     0   2719
Yemen             128   211   160   174   217   5744
Zambia             60   102    69    46    59   4392
Zimbabwe          508   494   434   437   407  11313

[195 rows x 38 columns]
```

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

[26]:
```
# 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
Total                 30449
Name: Japan, dtype: object
```

[27]:
```
# 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
982
```

[ ]:
```
# 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]])
```

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'.

[28]:
```
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:

[74]:
```
# useful for plotting later on
years = list(map(str, range(1980, 2014)))
years
```

[74]:
```
['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']
```

2.3.3 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).

[29]:
```
# 1. create the condition boolean series
condition = df_can['Continent'] == 'Asia'
print(condition)
```

```
Afghanistan         True
Albania            False
Algeria            False
American Samoa     False
Andorra            False
                   …
Viet Nam            True
Western Sahara     False
Yemen               True
Zambia             False
Zimbabwe           False
Name: Continent, Length: 195, dtype: bool
```

[30]:
```
# 2. pass this condition into the dataFrame
df_can[condition]
```

[30]:

	Continent	Region
Afghanistan	Asia	Southern Asia
Armenia	Asia	Western Asia
Azerbaijan	Asia	Western Asia
Bahrain	Asia	Western Asia
Bangladesh	Asia	Southern Asia
Bhutan	Asia	Southern Asia
Brunei Darussalam	Asia	South-Eastern Asia
Cambodia	Asia	South-Eastern Asia
China	Asia	Eastern Asia
China, Hong Kong Special Administrative Region	Asia	Eastern Asia
China, Macao Special Administrative Region	Asia	Eastern Asia
Cyprus	Asia	Western Asia
Democratic People's Republic of Korea	Asia	Eastern Asia
Georgia	Asia	Western Asia
India	Asia	Southern Asia
Indonesia	Asia	South-Eastern Asia
Iran (Islamic Republic of)	Asia	Southern Asia
Iraq	Asia	Western Asia
Israel	Asia	Western Asia
Japan	Asia	Eastern Asia
Jordan	Asia	Western Asia
Kazakhstan	Asia	Central Asia

Kuwait	Asia	Western Asia	
Kyrgyzstan	Asia	Central Asia	
Lao People's Democratic Republic	Asia	South-Eastern Asia	
Lebanon	Asia	Western Asia	
Malaysia	Asia	South-Eastern Asia	
Maldives	Asia	Southern Asia	
Mongolia	Asia	Eastern Asia	
Myanmar	Asia	South-Eastern Asia	
Nepal	Asia	Southern Asia	
Oman	Asia	Western Asia	
Pakistan	Asia	Southern Asia	
Philippines	Asia	South-Eastern Asia	
Qatar	Asia	Western Asia	
Republic of Korea	Asia	Eastern Asia	
Saudi Arabia	Asia	Western Asia	
Singapore	Asia	South-Eastern Asia	
Sri Lanka	Asia	Southern Asia	
State of Palestine	Asia	Western Asia	
Syrian Arab Republic	Asia	Western Asia	
Tajikistan	Asia	Central Asia	
Thailand	Asia	South-Eastern Asia	
Turkey	Asia	Western Asia	
Turkmenistan	Asia	Central Asia	
United Arab Emirates	Asia	Western Asia	
Uzbekistan	Asia	Central Asia	
Viet Nam	Asia	South-Eastern Asia	
Yemen	Asia	Western Asia	

	DevName	1980	\
Afghanistan	Developing regions	16	
Armenia	Developing regions	0	
Azerbaijan	Developing regions	0	
Bahrain	Developing regions	0	
Bangladesh	Developing regions	83	
Bhutan	Developing regions	0	
Brunei Darussalam	Developing regions	79	
Cambodia	Developing regions	12	
China	Developing regions	5123	
China, Hong Kong Special Administrative Region	Developing regions	0	
China, Macao Special Administrative Region	Developing regions	0	
Cyprus	Developing regions	132	
Democratic People's Republic of Korea	Developing regions	1	
Georgia	Developing regions	0	
India	Developing regions	8880	
Indonesia	Developing regions	186	
Iran (Islamic Republic of)	Developing regions	1172	
Iraq	Developing regions	262	

							\
Israel	Developing regions	1403					
Japan	Developed regions	701					
Jordan	Developing regions	177					
Kazakhstan	Developing regions	0					
Kuwait	Developing regions	1					
Kyrgyzstan	Developing regions	0					
Lao People's Democratic Republic	Developing regions	11					
Lebanon	Developing regions	1409					
Malaysia	Developing regions	786					
Maldives	Developing regions	0					
Mongolia	Developing regions	0					
Myanmar	Developing regions	80					
Nepal	Developing regions	1					
Oman	Developing regions	0					
Pakistan	Developing regions	978					
Philippines	Developing regions	6051					
Qatar	Developing regions	0					
Republic of Korea	Developing regions	1011					
Saudi Arabia	Developing regions	0					
Singapore	Developing regions	241					
Sri Lanka	Developing regions	185					
State of Palestine	Developing regions	0					
Syrian Arab Republic	Developing regions	315					
Tajikistan	Developing regions	0					
Thailand	Developing regions	56					
Turkey	Developing regions	481					
Turkmenistan	Developing regions	0					
United Arab Emirates	Developing regions	0					
Uzbekistan	Developing regions	0					
Viet Nam	Developing regions	1191					
Yemen	Developing regions	1					

	1981	1982	1983	1984	1985	\
Afghanistan	39	39	47	71	340	
Armenia	0	0	0	0	0	
Azerbaijan	0	0	0	0	0	
Bahrain	2	1	1	1	3	
Bangladesh	84	86	81	98	92	
Bhutan	0	0	0	1	0	
Brunei Darussalam	6	8	2	2	4	
Cambodia	19	26	33	10	7	
China	6682	3308	1863	1527	1816	
China, Hong Kong Special Administrative Region	0	0	0	0	0	
China, Macao Special Administrative Region	0	0	0	0	0	
Cyprus	128	84	46	46	43	
Democratic People's Republic of Korea	1	3	1	4	3	
Georgia	0	0	0	0	0	

India	8670	8147	7338	5704	4211
Indonesia	178	252	115	123	100
Iran (Islamic Republic of)	1429	1822	1592	1977	1648
Iraq	245	260	380	428	231
Israel	1711	1334	541	446	680
Japan	756	598	309	246	198
Jordan	160	155	113	102	179
Kazakhstan	0	0	0	0	0
Kuwait	0	8	2	1	4
Kyrgyzstan	0	0	0	0	0
Lao People's Democratic Republic	6	16	16	7	17
Lebanon	1119	1159	789	1253	1683
Malaysia	816	813	448	384	374
Maldives	0	0	1	0	0
Mongolia	0	0	0	0	0
Myanmar	62	46	31	41	23
Nepal	1	6	1	2	4
Oman	0	0	8	0	0
Pakistan	972	1201	900	668	514
Philippines	5921	5249	4562	3801	3150
Qatar	0	0	0	0	0
Republic of Korea	1456	1572	1081	847	962
Saudi Arabia	0	1	4	1	2
Singapore	301	337	169	128	139
Sri Lanka	371	290	197	1086	845
State of Palestine	0	0	0	0	0
Syrian Arab Republic	419	409	269	264	385
Tajikistan	0	0	0	0	0
Thailand	53	113	65	82	66
Turkey	874	706	280	338	202
Turkmenistan	0	0	0	0	0
United Arab Emirates	2	2	1	2	0
Uzbekistan	0	0	0	0	0
Viet Nam	1829	2162	3404	7583	5907
Yemen	2	1	6	0	18

	1986	…	2005	2006	\
Afghanistan	496	…	3436	3009	
Armenia	0	…	224	218	
Azerbaijan	0	…	359	236	
Bahrain	0	…	12	12	
Bangladesh	486	…	4171	4014	
Bhutan	0	…	5	10	
Brunei Darussalam	12	…	4	5	
Cambodia	8	…	370	529	
China	1960	…	42584	33518	
China, Hong Kong Special Administrative Region	0	…	729	712	

China, Macao Special Administrative Region	0	…	21	32	
Cyprus	48	…	7	9	
Democratic People's Republic of Korea	0	…	14	10	
Georgia	0	…	114	125	
India	7150	…	36210	33848	
Indonesia	127	…	632	613	
Iran (Islamic Republic of)	1794	…	5837	7480	
Iraq	265	…	2226	1788	
Israel	1212	…	2446	2625	
Japan	248	…	1067	1212	
Jordan	181	…	1940	1827	
Kazakhstan	0	…	506	408	
Kuwait	4	…	66	35	
Kyrgyzstan	0	…	173	161	
Lao People's Democratic Republic	21	…	42	74	
Lebanon	2576	…	3709	3802	
Malaysia	425	…	593	580	
Maldives	0	…	0	0	
Mongolia	0	…	59	64	
Myanmar	18	…	210	953	
Nepal	13	…	607	540	
Oman	0	…	14	18	
Pakistan	691	…	14314	13127	
Philippines	4166	…	18139	18400	
Qatar	1	…	11	2	
Republic of Korea	1208	…	5832	6215	
Saudi Arabia	5	…	198	252	
Singapore	205	…	392	298	
Sri Lanka	1838	…	4930	4714	
State of Palestine	0	…	453	627	
Syrian Arab Republic	493	…	1458	1145	
Tajikistan	0	…	85	46	
Thailand	78	…	575	500	
Turkey	257	…	2065	1638	
Turkmenistan	0	…	40	26	
United Arab Emirates	5	…	31	42	
Uzbekistan	0	…	330	262	
Viet Nam	2741	…	1852	3153	
Yemen	7	…	161	140	

	2007	2008	2009	2010	\
Afghanistan	2652	2111	1746	1758	
Armenia	198	205	267	252	
Azerbaijan	203	125	165	209	
Bahrain	22	9	35	28	
Bangladesh	2897	2939	2104	4721	
Bhutan	7	36	865	1464	

Brunei Darussalam	11	10	5	12
Cambodia	460	354	203	200
China	27642	30037	29622	30391
China, Hong Kong Special Administrative Region	674	897	657	623
China, Macao Special Administrative Region	16	12	21	21
Cyprus	4	7	6	18
Democratic People's Republic of Korea	7	19	11	45
Georgia	132	112	128	126
India	28742	28261	29456	34235
Indonesia	657	661	504	712
Iran (Islamic Republic of)	6974	6475	6580	7477
Iraq	2406	3543	5450	5941
Israel	2401	2562	2316	2755
Japan	1250	1284	1194	1168
Jordan	1421	1581	1235	1831
Kazakhstan	436	394	431	377
Kuwait	62	53	68	67
Kyrgyzstan	135	168	173	157
Lao People's Democratic Republic	53	32	39	54
Lebanon	3467	3566	3077	3432
Malaysia	600	658	640	802
Maldives	2	1	7	4
Mongolia	82	59	118	169
Myanmar	1887	975	1153	556
Nepal	511	581	561	1392
Oman	16	10	7	14
Pakistan	10124	8994	7217	6811
Philippines	19837	24887	28573	38617
Qatar	5	9	6	18
Republic of Korea	5920	7294	5874	5537
Saudi Arabia	188	249	246	330
Singapore	690	734	366	805
Sri Lanka	4123	4756	4547	4422
State of Palestine	441	481	400	654
Syrian Arab Republic	1056	919	917	1039
Tajikistan	44	15	50	52
Thailand	487	519	512	499
Turkey	1463	1122	1238	1492
Turkmenistan	37	13	20	30
United Arab Emirates	37	33	37	86
Uzbekistan	284	215	288	289
Viet Nam	2574	1784	2171	1942
Yemen	122	133	128	211

	2011	2012	2013	Total
Afghanistan	2203	2635	2004	65977
Armenia	236	258	207	6069

Azerbaijan	138	161	57	5408
Bahrain	21	39	32	3234
Bangladesh	2694	2640	3789	72906
Bhutan	1879	1075	487	13214
Brunei Darussalam	6	3	6	3357
Cambodia	196	233	288	9295
China	28502	33024	34129	662705
China, Hong Kong Special Administrative Region	591	728	774	12070
China, Macao Special Administrative Region	13	33	29	3027
Cyprus	6	12	16	3885
Democratic People's Republic of Korea	97	66	17	3131
Georgia	139	147	125	4827
India	27509	30933	33087	699242
Indonesia	390	395	387	15907
Iran (Islamic Republic of)	7479	7534	11291	183261
Iraq	6196	4041	4918	72548
Israel	1970	2134	1945	69267
Japan	1265	1214	982	30449
Jordan	1635	1206	1255	38165
Kazakhstan	381	462	348	15827
Kuwait	58	73	48	4784
Kyrgyzstan	159	278	123	9690
Lao People's Democratic Republic	22	25	15	3846
Lebanon	3072	1614	2172	118118
Malaysia	409	358	204	27174
Maldives	3	1	1	7368
Mongolia	103	68	99	3695
Myanmar	368	193	262	12002
Nepal	1129	1185	1308	17560
Oman	10	13	11	2983
Pakistan	7468	11227	12603	248938
Philippines	36765	34315	29544	514148
Qatar	3	14	6	2916
Republic of Korea	4588	5316	4509	145324
Saudi Arabia	278	286	267	6184
Singapore	219	146	141	17336
Sri Lanka	3309	3338	2394	155696
State of Palestine	555	533	462	9271
Syrian Arab Republic	1005	650	1009	34244
Tajikistan	47	34	39	7840
Thailand	396	296	400	11931
Turkey	1257	1068	729	34540
Turkmenistan	20	20	14	7647
United Arab Emirates	60	54	46	3595
Uzbekistan	162	235	167	10705
Viet Nam	1723	1731	2112	99903
Yemen	160	174	217	5744

[49 rows x 38 columns]

[31]:
```
# 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
```

[31]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	…	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	…	3436	3009	2652	2111	1746	1758	2203	2635	2004	65977
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	…	4171	4014	2897	2939	2104	4721	2694	2640		
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	0	…	5	10	7	36	865	1464	1879	1075		
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	…	36210	33848	28742	28261	29456	34235	27509	30933		
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	…	5837	7480	6974	6475	6580	7477	7479	7534		
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	…	0	0	2	1	7	4	3	1		
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	13	…	607	540	511	581	561	1392	1129	1185		
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	…	14314	13127	10124	8994	7217	6811	7468	11227		
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	…	4930	4714	4123	4756	4547	4422	3309	3338		

```
Bangladesh                    3789   72906
Bhutan                         487   13214
India                        33087  699242
Iran (Islamic Republic of)   11291  183261
Maldives                         1    7368
Nepal                         1308   17560
Pakistan                     12603  248938
Sri Lanka                     2394  155696

[9 rows x 38 columns]
```

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

[32]:
```
print('data dimensions:', df_can.shape)
print(df_can.columns)
df_can.head(2)
```

```
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')
```

[32]:
```
            Continent           Region             DevName  1980  1981  1982  \
Afghanistan      Asia    Southern Asia  Developing regions    16    39    39
Albania        Europe  Southern Europe   Developed regions     1     0     0

             1983  1984  1985  1986  …  2005  2006  2007  2008  2009  2010  \
Afghanistan    47    71   340   496  …  3436  3009  2652  2111  1746  1758
Albania         0     0     0     1  …  1223   856   702   560   716   561

             2011  2012  2013  Total
Afghanistan  2203  2635  2004  65977
Albania       539   620   603  18433

[2 rows x 38 columns]
```

---

3 Visualizing Data using Matplotlib

3.1 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.

3.1.1 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:

[33]:
```
# we are using the inline backend
%matplotlib inline

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

*optional: check if Matplotlib is loaded.

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

*optional: apply a style to Matplotlib.

[55]:
```
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-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-
darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper',
'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-
white', 'seaborn-whitegrid', 'tableau-colorblind10']
```

3.1.2 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

4 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.

[34]:
```
%matplotlib inline


import matplotlib as mpl
import matplotlib.pyplot as plt
years=list(map(str,range(1980,2014)))

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

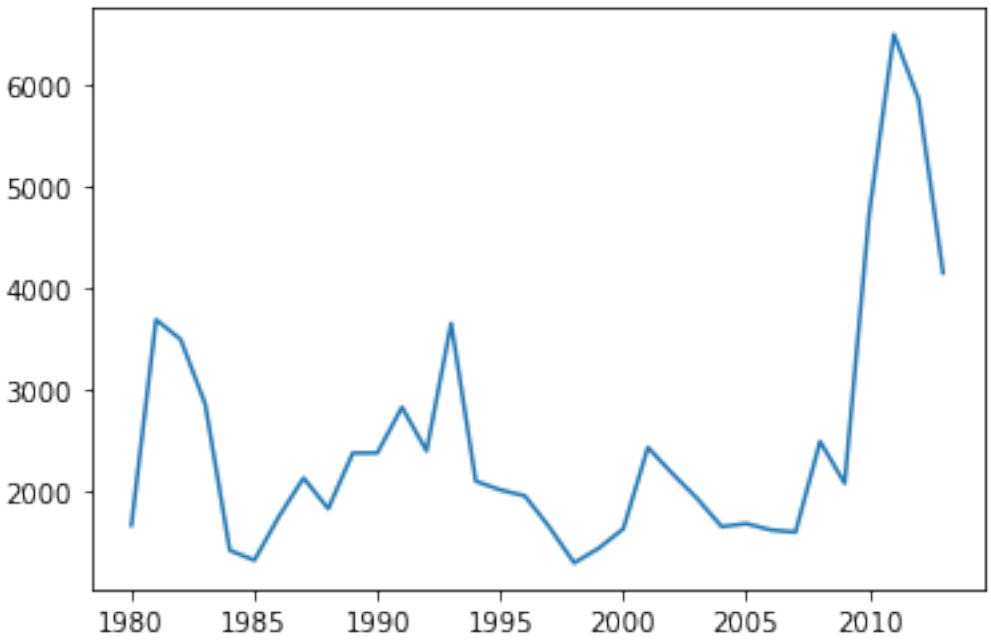
[34]:
```
1980    1666
1981    3692
1982    3498
1983    2860
1984    1418
Name: Haiti, dtype: object
```

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

[35]:
```
years=list(map(str,range(1980,2014)))

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

[35]:
```
<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.

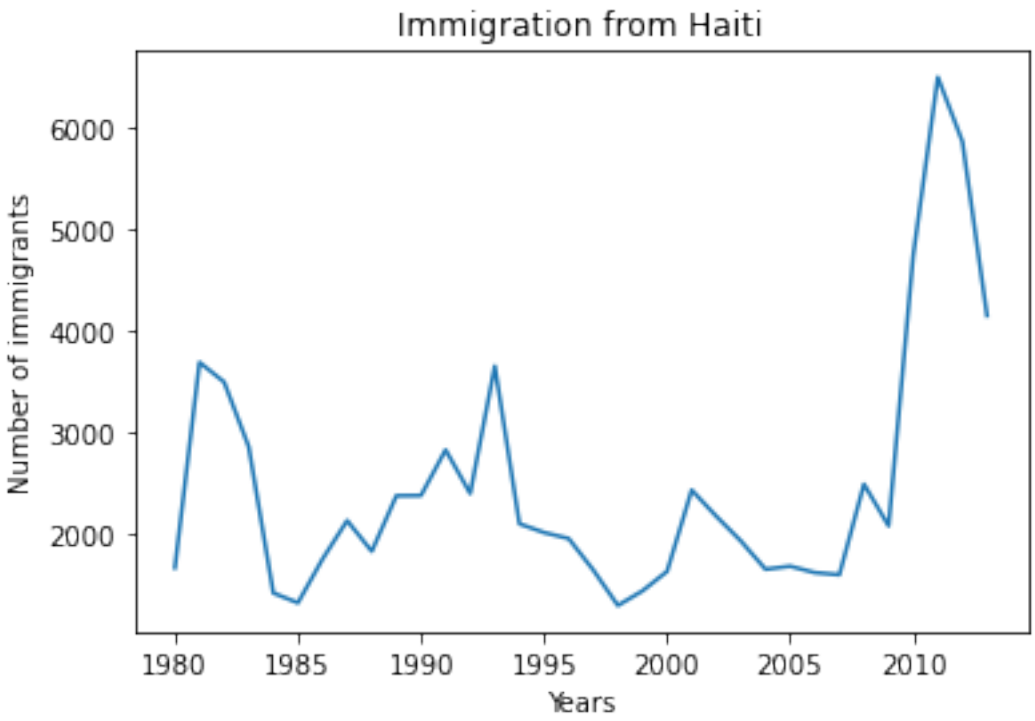
Also, let's label the x and y axis using `plt.title()`, `plt.ylabel()`, and `plt.xlabel()` as follows:

[36]:
```
years=list(map(str,range(1980,2014)))
haiti = df_can.loc['Haiti',years] # passing in years 1980 - 2013 to exclude the␣
 ↪'total' column

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
```



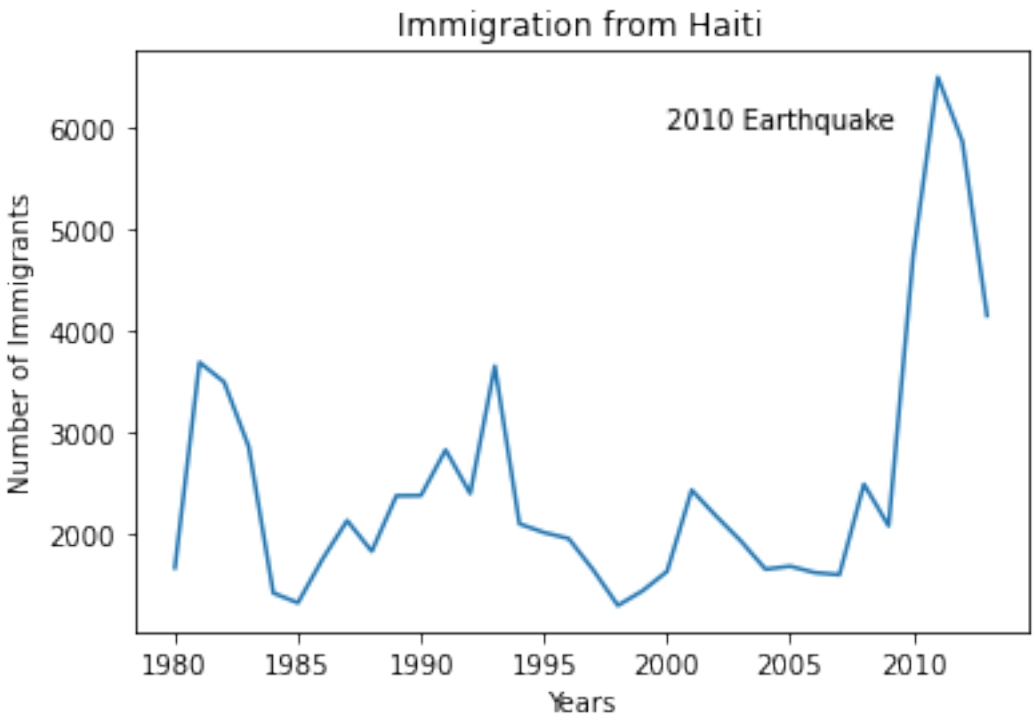
We can clearly notice how number of immigrants from Haiti spiked up from 2010 as Canada stepped up its efforts to accept refugees from Haiti. Let's annotate this spike in the plot by using the `plt.text()` method.

[37]:
```
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 i
    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
    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.

[40]:
```
### type your answer here
years=list(map(str,range(1980,2014)))
df_IC=df_can.loc[["India","China"],years]
df_IC.head()
```

[40]:

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	…	\
India	8880	8670	8147	7338	5704	4211	7150	10189	11522	10343	…	

```
China  5123  6682  3308  1863  1527  1816  1960   2643   2758   4323  …

        2004   2005   2006   2007   2008   2009   2010   2011   2012   2013
India  28235  36210  33848  28742  28261  29456  34235  27509  30933  33087
China  36619  42584  33518  27642  30037  29622  30391  28502  33024  34129

[2 rows x 34 columns]
```

Click here for a sample python solution

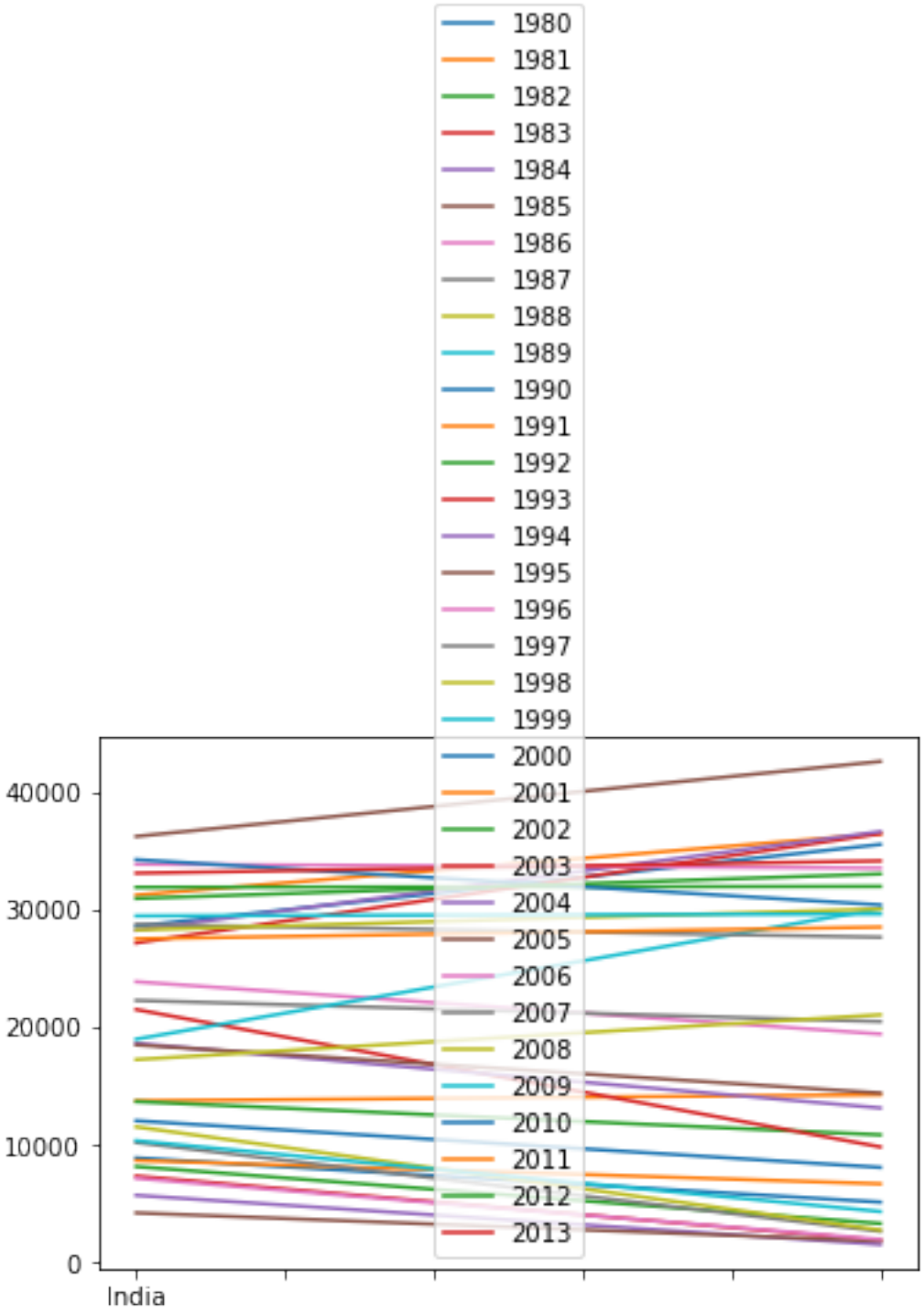
```
#The correct answer is:
df_CI = df_can.loc[['India', 'China'], years]
df_CI.head()
```

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

[42]:
```
### type your answer here

df_IC.plot(kind="line")
```

[42]:
```
<AxesSubplot:>
```



Click here for a sample python solution

```
#The correct answer is:
df_CI.plot(kind='line')
```

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.

[44]:
```
df_CI = df_IC.transpose()
df_CI.head()
```

[44]:

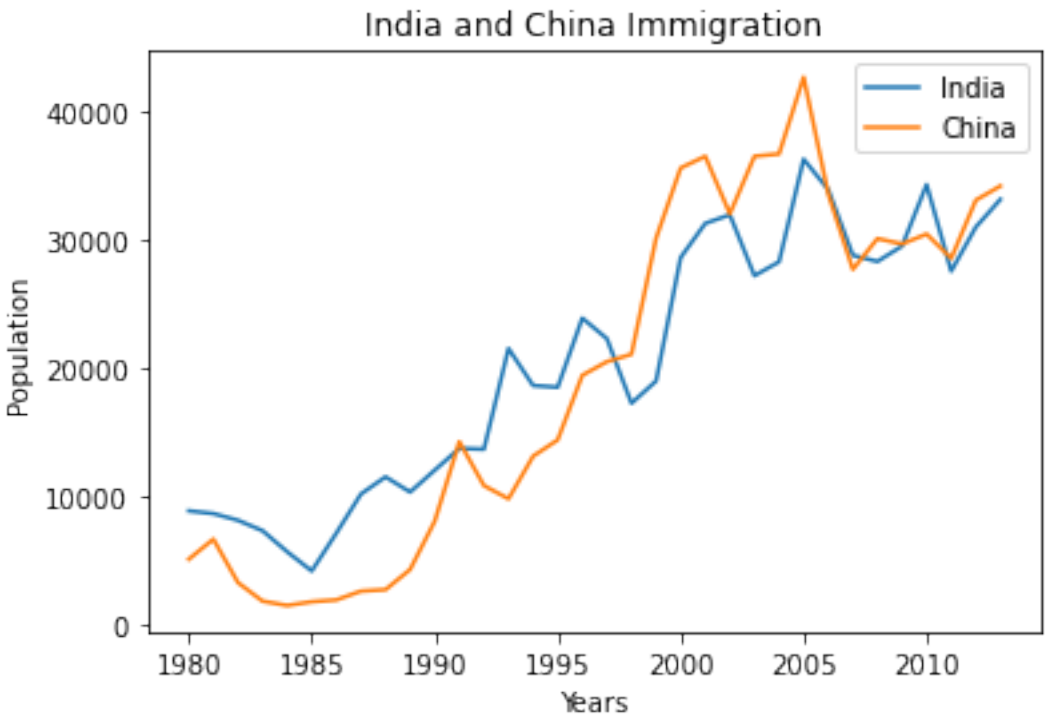
	India	China
1980	8880	5123
1981	8670	6682
1982	8147	3308
1983	7338	1863
1984	5704	1527

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.

[49]:
```
### type your answer here

df_CI.plot()
plt.title("India and China Immigration")
plt.xlabel("Years")
plt.ylabel("Population")
```

[49]:
```
Text(0, 0.5, 'Population')
```



Click here for a sample python solution

```
#The correct answer is:
df_CI.index = df_CI.index.map(int) # let's change the index values of df_CI to type intege
df_CI.plot(kind='line')

plt.title('Immigrants from China and India')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')

plt.show()
```

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))
```

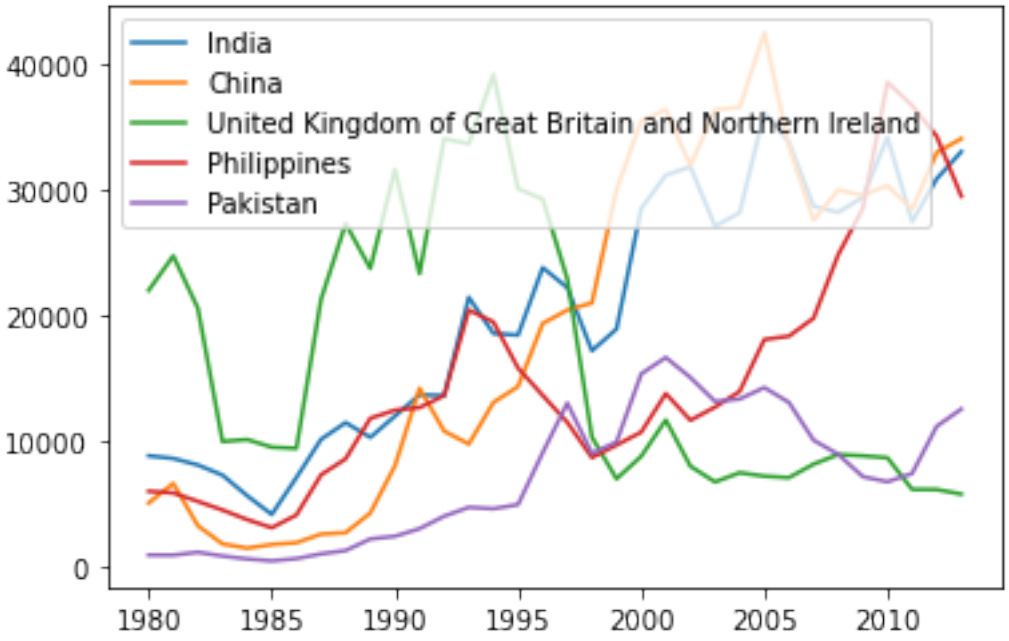
> class 'pandas.core.series.Series' 1980 1666 1981 3692 1982 3498 1983 2860 1984 1418
> Name: Haiti, dtype: int64

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.

[67]:
```
### type your answer here
df_can.sort_values(by='Total', ascending=False)
df_top5=df_can.head(5)
df_top5=df_top5[years].transpose()
#df_top5.drop(["Region","Continent","DevName"],axis=0)
#df_top5.index=df_top5.index.map(int)
df_top5.head()
df_top5.index=df_top5.index.map(int)
df_top5.plot()
```

[67]: <AxesSubplot:>



Click here for a sample python solution

```
#The correct answer is:
#Step 1: Get the dataset. Recall that we created a Total column that calculates cumulative
#We will sort on this column to get our top 5 countries using pandas sort_values() method.

inplace = True paramemter saves the changes to the original df_can dataframe
df_can.sort_values(by='Total', ascending=False, axis=0, inplace=True)

# get the top 5 entries
df_top5 = df_can.head(5)

# transpose the dataframe
df_top5 = df_top5[years].transpose()

print(df_top5)


#Step 2: Plot the dataframe. To make the plot more readeable, we will change the size usin
df_top5.index = df_top5.index.map(int) # let's change the index values of df_top5 to type
df_top5.plot(kind='line', figsize=(14, 8)) # pass a tuple (x, y) size



plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
```

```
plt.show()
```

4.0.1 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

4.0.2 Thank you for completing this lab!

4.1 Author

Alex Aklson

4.1.1 Other Contributors

Jay Rajasekharan Ehsan M. Kermani Slobodan Markovic.

4.2 Change Log

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

##