

# Introduction to Matplotlib and Line Plots

Estimated time needed: **20** minutes

## Objectives

After completing this lab you will be able to:

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

## Introduction

The aim of these labs is to introduce you to introduction you to Matplotlib and creating Line Plots. Please make sure that you have completed the prerequisites for this course, namely [Python Basics for Data Science](#) and [Analyzing Data with Python](#).

---

## Table of Contents

### *pandas* Refresher

The course heavily relies on *pandas* for data wrangling, analysis. Refresh your *Pandas* skill quickly with the lab on [Data pre-processing with Pandas](#)

*pandas* is an essential data analysis toolkit for Python.

We encourage you to spend some time and familiarize yourself with the *pandas* from the [website](#)

### The Dataset: Immigration to Canada from 1980 to 2013

Dataset Source: [International migration flows to and from selected countries - The 2015 revision](#). In this lab, we will focus on the Canadian immigration data. We have already **pre-processed** the data, we will use the **clean data** saved in the csv format for this lab. The Canada Immigration dataset can be fetched from [here](#).

---

Next, 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*'s `read_csv ()` method.

```
df_can = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.csv')
```

```
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)*

	Country	Continent	Region	DevName	1980
0	Afghanistan	Asia	Southern Asia	Developing regions	1639
1	Albania	Europe	Southern Europe	Developed regions	10
2	Algeria	Africa	Northern Africa	Developing regions	8067
3	American Samoa	Oceania	Polynesia	Developing regions	01
4	Andorra	Europe	Southern Europe	Developed regions	00

	1982	1983	1984	1985	...	2005	2006	2007	2008	2009	2010
0	39	47	71	340	...	3436	3009	2652	2111	1746	1758
1	0	0	0	0	...	1223	856	702	560	716	561
2	71	69	63	44	...	3626	4807	3623	4005	5393	4752
3	0	0	0	0	...	0	1	0	0	0	0
4	0	0	0	0	...	0	1	1	0	0	0

	2012	2013	Total
0	2635	2004	58639
1	620	603	15699
2	3774	4331	69439
3	0	0	6
4	1	1	15

[5 rows x 39 columns]

Let's set Country as the index, it will help you to plot the charts easily, by referring to the country names as index value

```
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()

#let's check
df_can.head(3)
```

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

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

	2011	2012	2013	Total
Country				
Afghanistan	2203	2635	2004	58639
Albania	539	620	603	15699
Algeria	4325	3774	4331	69439

[3 rows x 38 columns]

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

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:

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

```
[ '1980',  
  '1981',  
  '1982',  
  '1983',  
  '1984',  
  '1985',  
  '1986',  
  '1987',  
  '1988',  
  '1989',  
  '1990',  
  '1991',  
  '1992',  
  '1993',  
  '1994',  
  '1995',  
  '1996',  
  '1997',  
  '1998',  
  '1999',  
  '2000',  
  '2001',  
  '2002',  
  '2003',  
  '2004',  
  '2005',  
  '2006',  
  '2007',  
  '2008',  
  '2009',  
  '2010',  
  '2011',  
  '2012',  
  '2013']
```

---

# Visualizing Data using Matplotlib

## Matplotlib: Standard Python Visualization Library

The primary plotting library we will explore in the course is [Matplotlib](#). As mentioned on their website:

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

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

## Matplotlib.Pyplot

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

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

```
# 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

Matplotlib version: 3.5.3
```

\*optional: apply a style to Matplotlib.

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

['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-
gallery-nogrid', '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']
```

## Plotting in *pandas*

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

Documentation:

- [Plotting with Series](#)
- [Plotting with Dataframes](#)

## Line Plots (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:

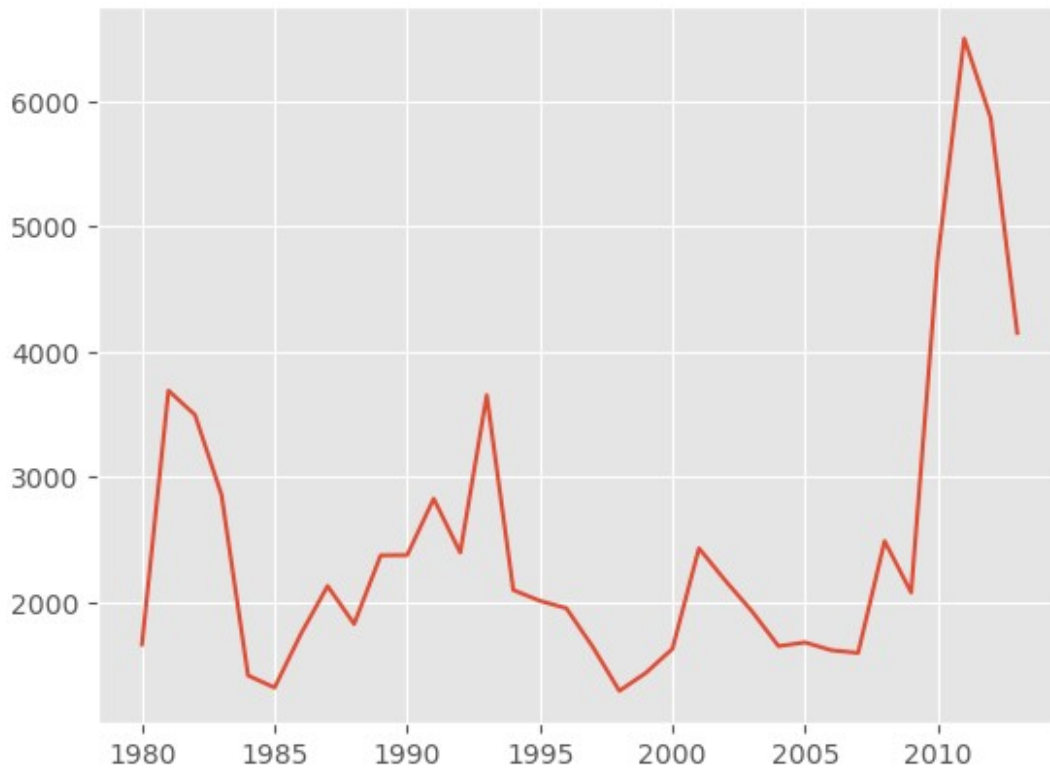
**Question:** Plot a line graph of immigration from Haiti using `df.plot()`.

First, we will extract the data series for Haiti.

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

```
haiti.plot()  
<AxesSubplot:>
```



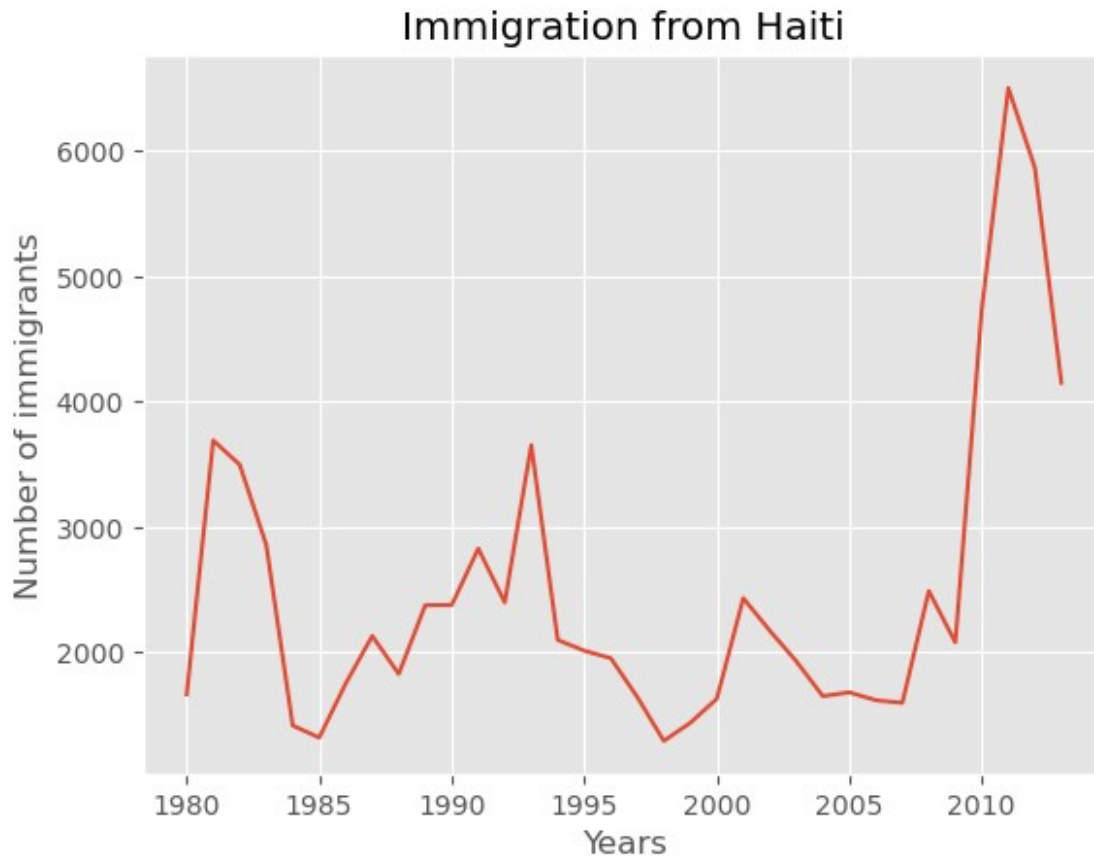
*pandas* automatically populated the x-axis with the index values (years), and the y-axis with the column values (population).

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

```
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. However, notice that years are of type *string*. Let's change the type of the index values to *integer* first.

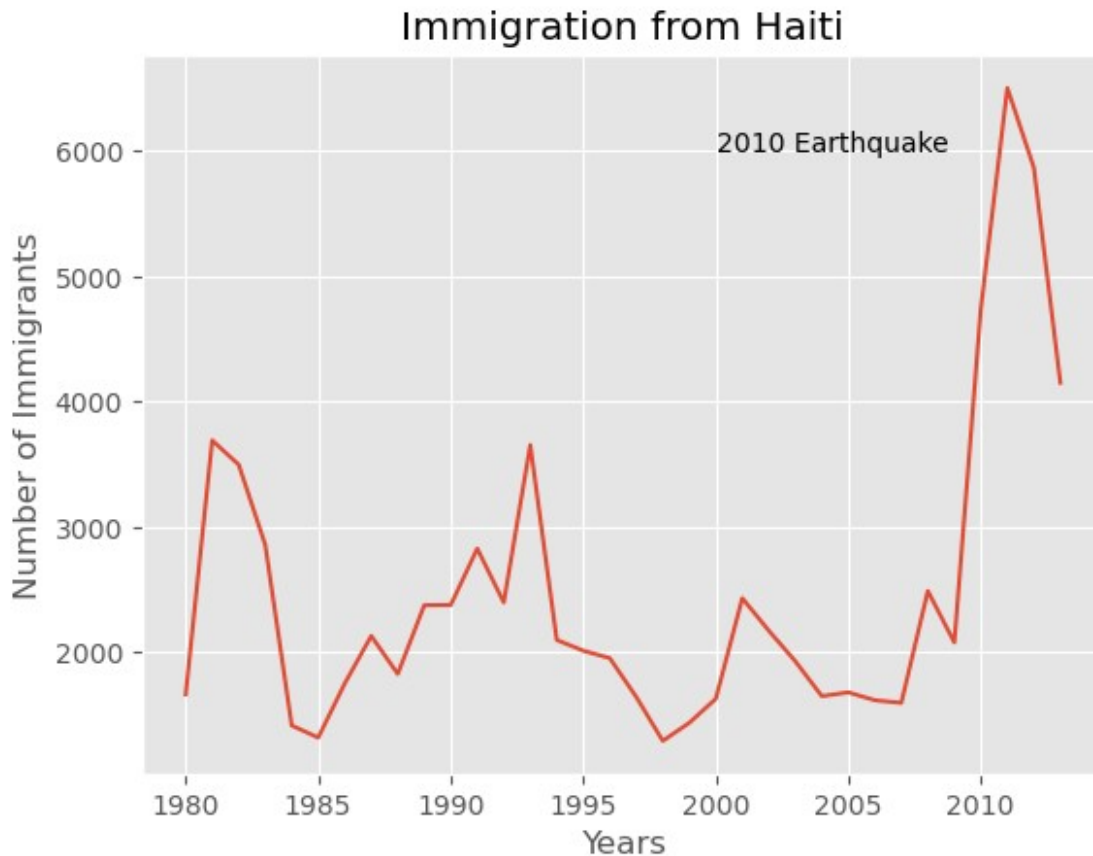
```
haiti.index = haiti.index.map(int)
haiti.plot(kind='line')

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

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

plt.show()
```





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

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

Since the x-axis (years) is type 'integer', we specified x as a year. The y axis (number of immigrants) is type 'integer', so we can just specify the value y = 6000.

```
plt.text(2000, 6000, '2010 Earthquake') # years stored as type int
```

If the years were stored as type 'string', we would need to specify x as the index position of the year. Eg 20th index is year 2000 since it is the 20th year with a base year of 1980.

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

We will cover advanced annotation methods in later modules.

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

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

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

```
### type your answer here
```

```
df_CI = df_can.loc[['India', 'China'], years]
df_CI
```

	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	...								
	2003	2004	2005	2006	2007	2008	2009	2010	2011
2012									
India	27155	28235	36210	33848	28742	28261	29456	34235	27509
30933									
China	36439	36619	42584	33518	27642	30037	29622	30391	28502
33024									

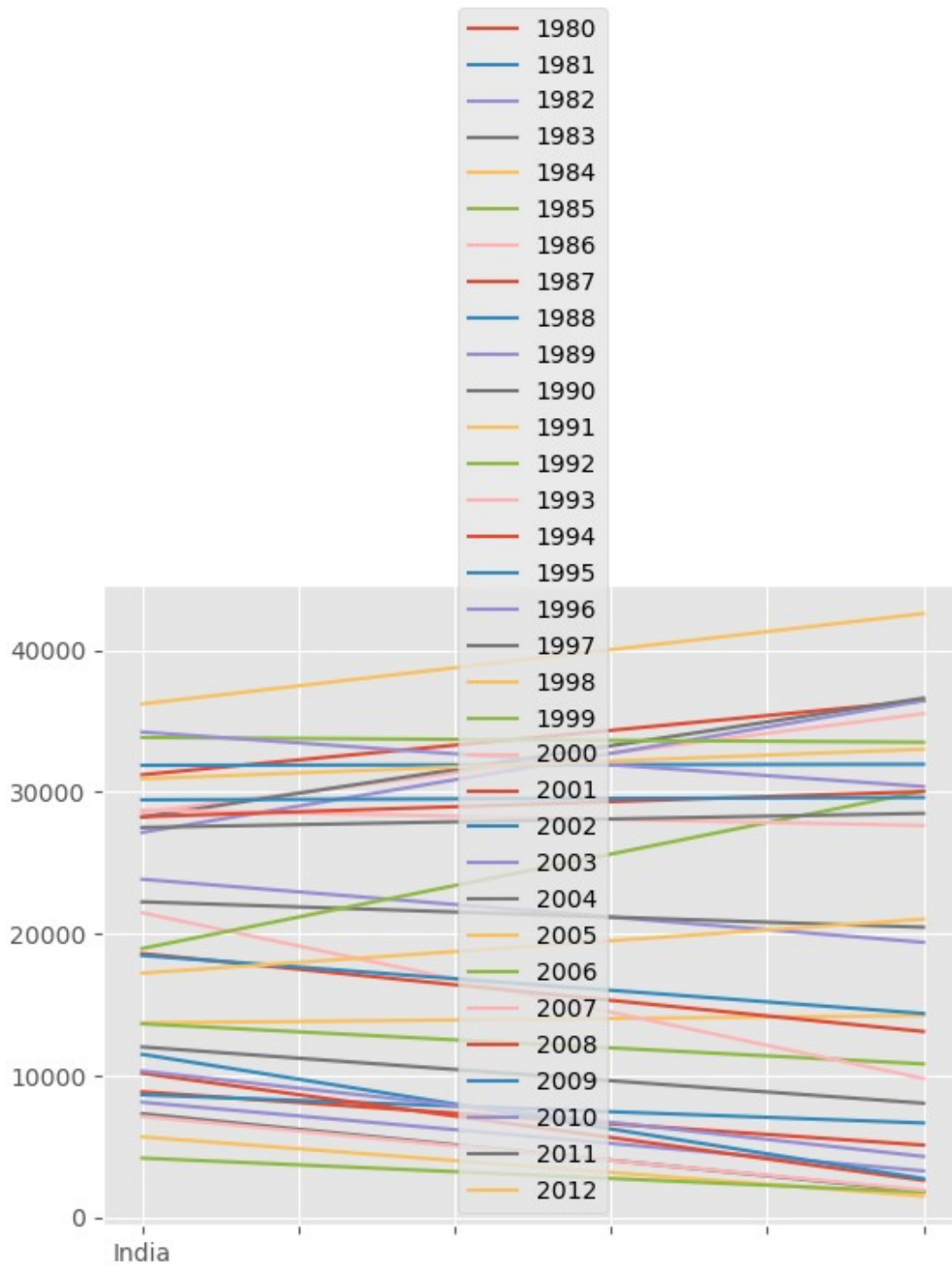
```
[2 rows x 33 columns]
```

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

```
### type your answer here
```

```
df_CI.plot(kind='line')
```

```
<AxesSubplot:>
```



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.

```
df_CI = df_CI.transpose()
df_CI.head()
```

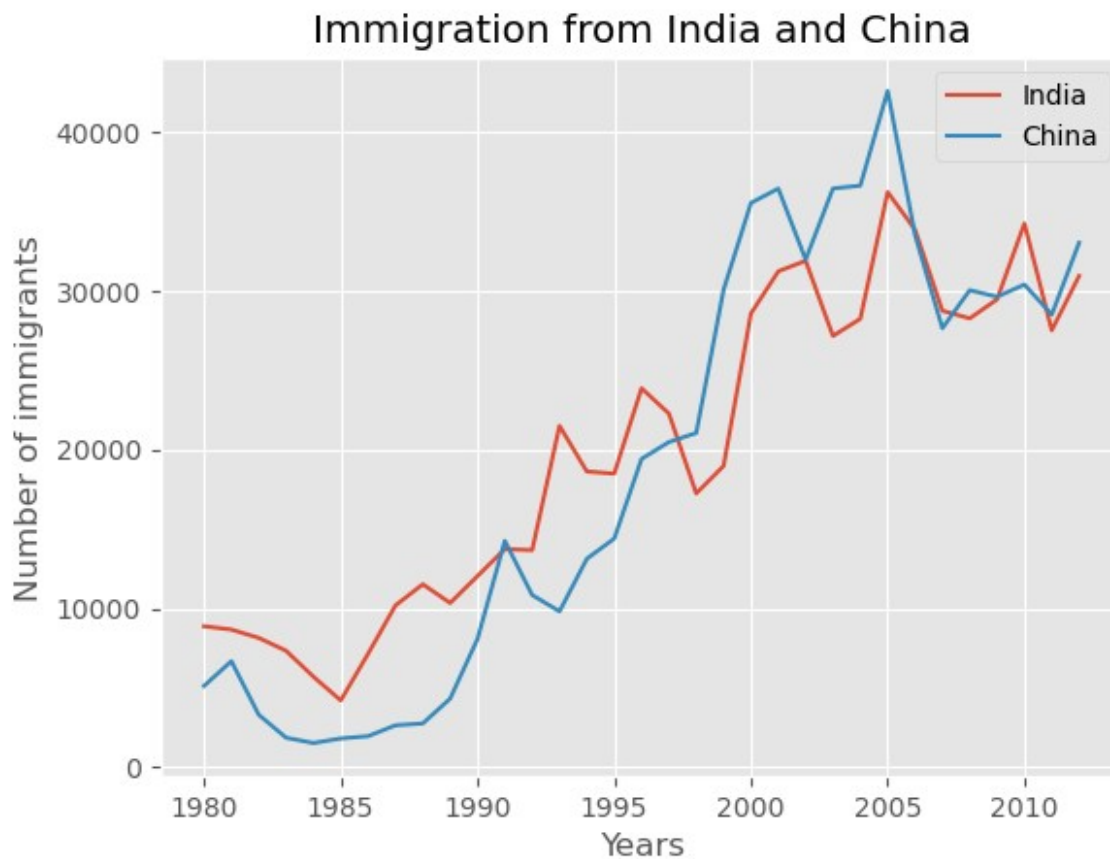
	India	China
1980	8880	5123
1981	8670	6682
1982	8147	3308
1983	7338	1863
1984	5704	1527

*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.

```
### type your answer here
df_CI.index = df_CI.index.map(int)
df_CI.plot(kind='line')

plt.title('Immigration from India and China')
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.

```
### type your answer here
inplace = True # parameter saves the changes to the original df_can
dataframe
df_can.sort_values(by='Total', ascending=False, axis=0, inplace=True)

df_top5 = df_can.head(5)

df_top5 = df_top5[years].transpose()

print(df_top5)

df_top5.index = df_top5.index.map(int)
df_top5.plot(kind='line', figsize=(14, 8))

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

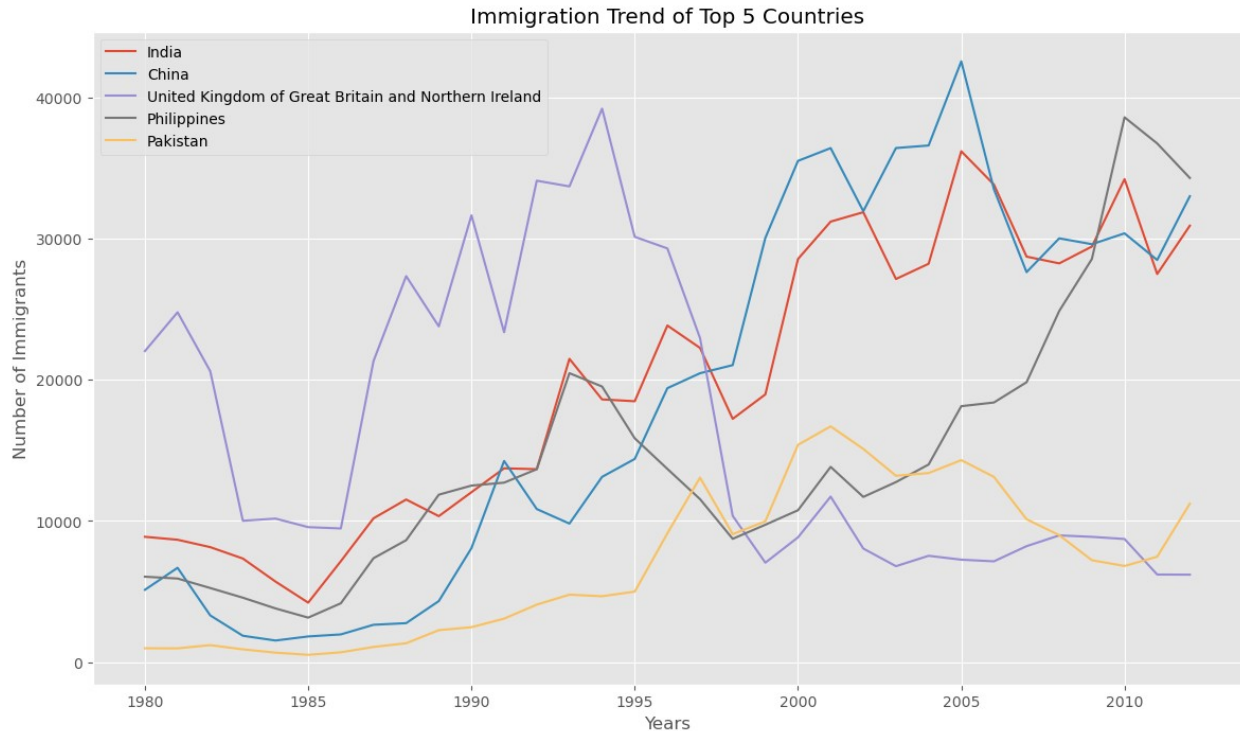
plt.show()
```

	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

	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



## Other Plots

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

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

Thank you for completing this lab!

## Author

Alex Aklson

## Other Contributors

Jay Rajasekharan, Ehsan M. Kermani, Slobodan Markovic, Weiqing Wang, Dr. Pooja



## Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-06-08	2.5	Dr. Pooja	Updated to work with clean data csv file
2021-05-29	2.4	Weiqing Wang	Fixed typos and code spells.
2021-01-20	2.3	Lakshmi Holla	Changed TOC cell markdown
2020-11-20	2.2	Lakshmi Holla	Changed IBM box URL
2020-11-03	2.1	Lakshmi Holla	Changed URL and info method
2020-08-27	2.0	Lavanya	Moved Lab to course repo in GitLab

© IBM Corporation 2020. All rights reserved.