
# Data Visualization

Estimated time needed: **30** minutes

## Objectives

After completing this lab you will be able to:

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

# Introduction

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

Please make sure that you have completed the prerequisites for this course, namely **Python Basics for Data Science** (https://www.edx.org/course/python-basics-for-data-science-2?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) and **Analyzing Data with Python** (https://www.edx.org/course/data-analysis-with-python?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

**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** (https://www.edx.org/course/data-analysis-with-python?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

---

# Table of Contents

# Exploring Datasets with *pandas*

*pandas* is an essential data analysis toolkit for Python. From their website (http://pandas.pydata.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01):

> *pandas* is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python.

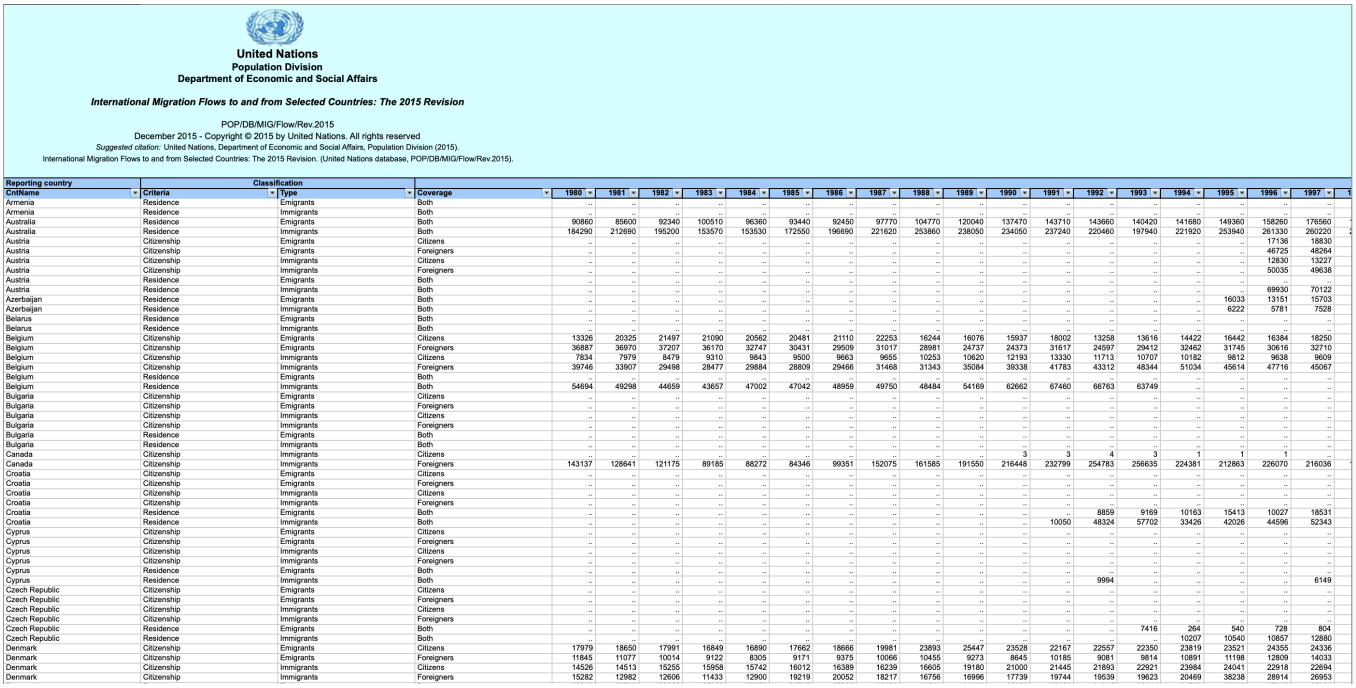
The course heavily relies on *pandas* for data wrangling, analysis, and visualization. We encourage you to spend some time and familiarize yourself with the *pandas* API Reference: http://pandas.pydata.org/pandas-docs/stable/api.html (http://pandas.pydata.org/pandas-docs/stable/api.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

## 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?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01)](http://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.shtml?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

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 | Classification | | | 1980 | 1981 | 1982 | 1983 | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CntName | Criteria | Type | Coverage | | | | | | | | | | | | | | | | | | |
| Armenia | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Armenia | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Australia | Residence | Emigrants | Both | 90860 | 85600 | 92340 | 100510 | 96360 | 93440 | 92450 | 97770 | 104770 | 120040 | 137470 | 143710 | 143660 | 140420 | 141680 | 149360 | 158260 | 176560 |
| Australia | Residence | Immigrants | Both | 184290 | 212690 | 195200 | 153570 | 153530 | 172550 | 196690 | 221620 | 253860 | 238050 | 234050 | 237240 | 220460 | 197940 | 221920 | 253940 | 261330 | 260220 |
| Austria | Citizenship | Emigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 17136 | 18830 |
| Austria | Citizenship | Emigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 46725 | 48264 |
| Austria | Citizenship | Immigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 12830 | 13227 |
| Austria | Citizenship | Immigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 50035 | 49638 |
| Austria | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Austria | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 69930 | 70122 |
| Azerbaijan | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 16033 | 13151 | 15703 |
| Azerbaijan | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 6222 | 5781 | 7528 |
| Belarus | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Belarus | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Belgium | Citizenship | Emigrants | Citizens | 13326 | 20325 | 21497 | 21090 | 20562 | 20481 | 21110 | 22253 | 16244 | 16076 | 15937 | 18002 | 13258 | 13616 | 14422 | 16442 | 16384 | 18250 |
| Belgium | Citizenship | Emigrants | Foreigners | 36887 | 36970 | 37207 | 36170 | 32747 | 30431 | 29509 | 31017 | 28981 | 24737 | 24373 | 31617 | 24597 | 29412 | 32462 | 31745 | 30616 | 32710 |
| Belgium | Citizenship | Immigrants | Citizens | 7834 | 7979 | 8479 | 9310 | 9843 | 9500 | 9663 | 9655 | 10253 | 10620 | 12193 | 13330 | 11713 | 10707 | 10182 | 9812 | 9638 | 9609 |
| Belgium | Citizenship | Immigrants | Foreigners | 39746 | 33907 | 29498 | 28477 | 29884 | 28809 | 29466 | 31468 | 31343 | 35084 | 39338 | 41783 | 43312 | 48344 | 51034 | 45614 | 47716 | 45067 |
| Belgium | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Belgium | Residence | Immigrants | Both | 54694 | 49298 | 44659 | 43657 | 47002 | 47042 | 48959 | 49750 | 48484 | 54169 | 62662 | 67460 | 66763 | 63749 | .. | .. | .. | .. |
| Bulgaria | Citizenship | Emigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Bulgaria | Citizenship | Emigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Bulgaria | Citizenship | Immigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Bulgaria | Citizenship | Immigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Bulgaria | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Bulgaria | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Canada | Citizenship | Immigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 3 | 3 | 4 | 3 | 1 | 1 | .. | .. |
| Canada | Citizenship | Immigrants | Foreigners | 143137 | 128641 | 121175 | 89185 | 88272 | 84346 | 99351 | 152075 | 161585 | 191550 | 216448 | 232799 | 254783 | 256635 | 224381 | 212863 | 226070 | 216036 |
| Croatia | Citizenship | Emigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Croatia | Citizenship | Emigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Croatia | Citizenship | Immigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Croatia | Citizenship | Immigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Croatia | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 8859 | 9169 | 10163 | 15413 | 10027 | 18531 |
| Croatia | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 10050 | 48324 | 57702 | 33426 | 42026 | 44596 | 52343 |
| Cyprus | Citizenship | Emigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Cyprus | Citizenship | Emigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Cyprus | Citizenship | Immigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Cyprus | Citizenship | Immigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Cyprus | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Cyprus | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 9994 | .. | .. | .. | .. | 6149 |
| Czech Republic | Citizenship | Emigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Czech Republic | Citizenship | Emigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Czech Republic | Citizenship | Immigrants | Citizens | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Czech Republic | Citizenship | Immigrants | Foreigners | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| Czech Republic | Residence | Emigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 7416 | 264 | 540 | 728 | 804 |
| Czech Republic | Residence | Immigrants | Both | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | 10207 | 10540 | 10857 | 12880 |
| Denmark | Citizenship | Emigrants | Citizens | 17979 | 18650 | 17991 | 16849 | 16890 | 17662 | 18666 | 19981 | 23893 | 25447 | 23528 | 22167 | 22557 | 22350 | 23819 | 23521 | 24355 | 24336 |
| Denmark | Citizenship | Emigrants | Foreigners | 11845 | 11077 | 10014 | 9122 | 8305 | 9171 | 9375 | 10066 | 10455 | 9273 | 8645 | 10185 | 9081 | 9814 | 10891 | 11198 | 12809 | 14033 |
| Denmark | Citizenship | Immigrants | Citizens | 14526 | 14513 | 15255 | 15958 | 15742 | 16012 | 16389 | 16239 | 16605 | 19180 | 21000 | 21445 | 21893 | 22921 | 23984 | 24041 | 22918 | 22694 |
| Denmark | Citizenship | Immigrants | Foreigners | 15282 | 12982 | 12606 | 11433 | 12900 | 19219 | 20052 | 18217 | 16756 | 16996 | 17739 | 19744 | 19539 | 19623 | 20469 | 38238 | 28914 | 26953 |

The Canada Immigration dataset can be fetched from [here (https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.xlsx?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01)](https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.xlsx?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

# *pandas* Basics

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

In [1]:

```
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
```

Let's download and import our primary Canadian Immigration dataset using *pandas*'s `read_excel()` method. Normally, before we can do that, we would need to download a module which *pandas* requires reading in Excel files. This module was **openpyxl** (formerlly **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 **openpyxl** module:

```
! pip3 install openpyxl
```

Now we are ready to read in our data.

In [5]:

```
#! pip3 install openpyxl
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.

In [6]:

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

Out[6]:

| | Type | Coverage | OdName | AREA | AreaName | REG | RegName | DEV | DevName |
|---|---|---|---|---|---|---|---|---|---|
| **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 view the bottom 5 rows of the dataset using the `tail()` function.

In [7]:

```
df_can.tail()
```

Out[7]:

| | Type | Coverage | OdName | AREA | AreaName | REG | RegName | DEV | DevName |
|---|---|---|---|---|---|---|---|---|---|
| **190** | Immigrants | Foreigners | Viet Nam | 935 | Asia | 920 | South-Eastern Asia | 902 | Developing regions |
| **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.

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

In [8]:

```
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 data frame's `columns` instance variable.

In [9]:

```
df_can.columns
```

Out[9]:

```
Index([    'Type', 'Coverage',   'OdName',     'AREA', 'AreaName',      'R
EG',
       'RegName',      'DEV',  'DevName',       1980,       1981,       1
982,
            1983,       1984,       1985,       1986,       1987,       1
988,
            1989,       1990,       1991,       1992,       1993,       1
994,
            1995,       1996,       1997,       1998,       1999,       2
000,
            2001,       2002,       2003,       2004,       2005,       2
006,
            2007,       2008,       2009,       2010,       2011,       2
012,
            2013],
      dtype='object')
```

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

In [10]:

```
df_can.index
```

Out[10]:

```
RangeIndex(start=0, stop=195, step=1)
```

Note: The default type of intance variables `index` and `columns` are **NOT** `list`.

In [11]:

```
print(type(df_can.columns))
print(type(df_can.index))
```

```
<class 'pandas.core.indexes.base.Index'>
<class 'pandas.core.indexes.range.RangeIndex'>
```

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

In [12]:

```
df_can.columns.tolist()
```

Out[12]:

```
['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]
```

In [13]:

```
df_can.index.tolist()
```

Out[13]:

```
[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]
```

In [14]:

```
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` instance variable of it.

In [15]:

```
# size of dataframe (rows, columns)
df_can.shape
```

Out[15]:

```
(195, 43)
```

**Note**: The main types stored in *pandas* objects are `float`, `int`, `bool`, `datetime64[ns]`, `datetime64[ns, tz]`, `timedelta[ns]`, `category`, 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 [16]:

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

Out[16]:

| | OdName | AreaName | RegName | DevName | 1980 | 1981 | 1982 | 1983 | 1984 | 1985 | ... | 2( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Afghanistan | Asia | Southern Asia | Developing regions | 16 | 39 | 39 | 47 | 71 | 340 | ... | 2( |
| **1** | Albania | Europe | Southern Europe | Developed regions | 1 | 0 | 0 | 0 | 0 | 0 | ... | 1( |

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 [17]:

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

Out[17]:

```
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 [18]:

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

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_l
auncher.py:1: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version this
will raise TypeError.  Select only valid columns before calling the reduct
ion.
  """Entry point for launching an IPython kernel.
```

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

In [19]:

```
df_can.isnull().sum()
```

Out[19]:

```
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 [20]:

```
df_can.describe()
```

Out[20]:

| | 1980 | 1981 | 1982 | 1983 | 1984 | 1985 |
|---|---|---|---|---|---|---|
| **count** | 195.000000 | 195.000000 | 195.000000 | 195.000000 | 195.000000 | 195.000000 |
| **mean** | 508.394872 | 566.989744 | 534.723077 | 387.435897 | 376.497436 | 358.861538 |
| **std** | 1949.588546 | 2152.643752 | 1866.997511 | 1204.333597 | 1198.246371 | 1079.309600 |
| **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 |
| **max** | 22045.000000 | 24796.000000 | 20620.000000 | 10015.000000 | 10170.000000 | 9564.000000 |

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 [21]:

```
df_can.Country  # returns a series
```

Out[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 ('Country') and the data for years: 1980 - 1985.

In [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.
```

Out[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 × 7 columns

## Select Row

There are main 2 ways to select rows:

```
df.loc[label]    # filters by the labels of the index/column
    df.iloc[index]   # filters by the positions of the index/column
```

Before we proceed, notice that the default index of the dataset is a numeric range from 0 to 194. This makes it very difficult to do a query by a specific country. For example to search for data on Japan, we need to know the corresponding index value.

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

In [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_ind
ex()
```

In [24]:

```
df_can.head(3)
```

Out[24]:

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

3 rows × 38 columns

In [25]:

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

Example: Let's view the number of immigrants from Japan (row 87) for the following scenarios: 1. The full row data (all columns) 2. For year 2013 3. For years 1980 to 1985

In [26]:

```
# 1. the full row data (all columns)
df_can.loc['Japan']
```

Out[26]:

```
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 [27]:

```
# alternate methods
df_can.iloc[87]
```

Out[27]:

```
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 [28]:

```
df_can[df_can.index == 'Japan']
```

Out[28]:

| | Continent | Region | DevName | 1980 | 1981 | 1982 | 1983 | 1984 | 1985 | 1986 | ... | 2005 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Japan** | Asia | Eastern Asia | Developed regions | 701 | 756 | 598 | 309 | 246 | 198 | 248 | ... | 1067 |

1 rows × 38 columns

In [29]:

```
# 2. for year 2013
df_can.loc['Japan', 2013]
```

Out[29]:

```
982
```

In [30]:

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

Out[30]:

```
982
```

In [31]:

```
# 3. for years 1980 to 1985
df_can.loc['Japan', [1980, 1981, 1982, 1983, 1984, 1984]]
```

Out[31]:

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

In [32]:

```
# Alternative Method
df_can.iloc[87, [3, 4, 5, 6, 7, 8]]
```

Out[32]:

```
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 [33]:

```
df_can.columns = list(map(str, df_can.columns))
# [print (type(x)) for x in df_can.columns.values] #<-- uncomment to check type of colu
mn headers
```

Since we converted the years to string, let's declare a variable that will allow us to easily call upon the full range of years:

In [34]:

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

Out[34]:

```
['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 [35]:

```
# 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
```

In [36]:

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

Out[36]:

| | Continent | Region | DevName | 1980 | 1981 | 1982 | 1983 | 1984 | 1985 | 1986 | . |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Afghanistan** | Asia | Southern Asia | Developing regions | 16 | 39 | 39 | 47 | 71 | 340 | 496 | . |
| **Armenia** | Asia | Western Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Azerbaijan** | Asia | Western Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Bahrain** | Asia | Western Asia | Developing regions | 0 | 2 | 1 | 1 | 1 | 3 | 0 | . |
| **Bangladesh** | Asia | Southern Asia | Developing regions | 83 | 84 | 86 | 81 | 98 | 92 | 486 | . |
| **Bhutan** | Asia | Southern Asia | Developing regions | 0 | 0 | 0 | 0 | 1 | 0 | 0 | . |
| **Brunei Darussalam** | Asia | South-Eastern Asia | Developing regions | 79 | 6 | 8 | 2 | 2 | 4 | 12 | . |
| **Cambodia** | Asia | South-Eastern Asia | Developing regions | 12 | 19 | 26 | 33 | 10 | 7 | 8 | . |
| **China** | Asia | Eastern Asia | Developing regions | 5123 | 6682 | 3308 | 1863 | 1527 | 1816 | 1960 | . |
| **China, Hong Kong Special Administrative Region** | Asia | Eastern Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **China, Macao Special Administrative Region** | Asia | Eastern Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Cyprus** | Asia | Western Asia | Developing regions | 132 | 128 | 84 | 46 | 46 | 43 | 48 | . |
| **Democratic People's Republic of Korea** | Asia | Eastern Asia | Developing regions | 1 | 1 | 3 | 1 | 4 | 3 | 0 | . |
| **Georgia** | Asia | Western Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **India** | Asia | Southern Asia | Developing regions | 8880 | 8670 | 8147 | 7338 | 5704 | 4211 | 7150 | . |
| **Indonesia** | Asia | South-Eastern Asia | Developing regions | 186 | 178 | 252 | 115 | 123 | 100 | 127 | . |
| **Iran (Islamic Republic of)** | Asia | Southern Asia | Developing regions | 1172 | 1429 | 1822 | 1592 | 1977 | 1648 | 1794 | . |
| **Iraq** | Asia | Western Asia | Developing regions | 262 | 245 | 260 | 380 | 428 | 231 | 265 | . |
| **Israel** | Asia | Western Asia | Developing regions | 1403 | 1711 | 1334 | 541 | 446 | 680 | 1212 | . |
| **Japan** | Asia | Eastern Asia | Developed regions | 701 | 756 | 598 | 309 | 246 | 198 | 248 | . |
| **Jordan** | Asia | Western Asia | Developing regions | 177 | 160 | 155 | 113 | 102 | 179 | 181 | . |

| | Continent | Region | DevName | 1980 | 1981 | 1982 | 1983 | 1984 | 1985 | 1986 | . |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Kazakhstan** | Asia | Central Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Kuwait** | Asia | Western Asia | Developing regions | 1 | 0 | 8 | 2 | 1 | 4 | 4 | . |
| **Kyrgyzstan** | Asia | Central Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Lao People's Democratic Republic** | Asia | South-Eastern Asia | Developing regions | 11 | 6 | 16 | 16 | 7 | 17 | 21 | . |
| **Lebanon** | Asia | Western Asia | Developing regions | 1409 | 1119 | 1159 | 789 | 1253 | 1683 | 2576 | . |
| **Malaysia** | Asia | South-Eastern Asia | Developing regions | 786 | 816 | 813 | 448 | 384 | 374 | 425 | . |
| **Maldives** | Asia | Southern Asia | Developing regions | 0 | 0 | 0 | 1 | 0 | 0 | 0 | . |
| **Mongolia** | Asia | Eastern Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Myanmar** | Asia | South-Eastern Asia | Developing regions | 80 | 62 | 46 | 31 | 41 | 23 | 18 | . |
| **Nepal** | Asia | Southern Asia | Developing regions | 1 | 1 | 6 | 1 | 2 | 4 | 13 | . |
| **Oman** | Asia | Western Asia | Developing regions | 0 | 0 | 0 | 8 | 0 | 0 | 0 | . |
| **Pakistan** | Asia | Southern Asia | Developing regions | 978 | 972 | 1201 | 900 | 668 | 514 | 691 | . |
| **Philippines** | Asia | South-Eastern Asia | Developing regions | 6051 | 5921 | 5249 | 4562 | 3801 | 3150 | 4166 | . |
| **Qatar** | Asia | Western Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 1 | . |
| **Republic of Korea** | Asia | Eastern Asia | Developing regions | 1011 | 1456 | 1572 | 1081 | 847 | 962 | 1208 | . |
| **Saudi Arabia** | Asia | Western Asia | Developing regions | 0 | 0 | 1 | 4 | 1 | 2 | 5 | . |
| **Singapore** | Asia | South-Eastern Asia | Developing regions | 241 | 301 | 337 | 169 | 128 | 139 | 205 | . |
| **Sri Lanka** | Asia | Southern Asia | Developing regions | 185 | 371 | 290 | 197 | 1086 | 845 | 1838 | . |
| **State of Palestine** | Asia | Western Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Syrian Arab Republic** | Asia | Western Asia | Developing regions | 315 | 419 | 409 | 269 | 264 | 385 | 493 | . |
| **Tajikistan** | Asia | Central Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Thailand** | Asia | South-Eastern Asia | Developing regions | 56 | 53 | 113 | 65 | 82 | 66 | 78 | . |
| **Turkey** | Asia | Western Asia | Developing regions | 481 | 874 | 706 | 280 | 338 | 202 | 257 | . |

| | Continent | Region | DevName | 1980 | 1981 | 1982 | 1983 | 1984 | 1985 | 1986 | . |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Turkmenistan** | Asia | Central Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **United Arab Emirates** | Asia | Western Asia | Developing regions | 0 | 2 | 2 | 1 | 2 | 0 | 5 | . |
| **Uzbekistan** | Asia | Central Asia | Developing regions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | . |
| **Viet Nam** | Asia | South-Eastern Asia | Developing regions | 1191 | 1829 | 2162 | 3404 | 7583 | 5907 | 2741 | . |
| **Yemen** | Asia | Western Asia | Developing regions | 1 | 2 | 1 | 6 | 0 | 18 | 7 | . |

49 rows × 38 columns

In [37]:

```
# we can pass multiple criteria in the same line.
# let's filter for AreaNAme = Asia and RegName = Southern Asia

df_can[(df_can['Continent']=='Asia') & (df_can['Region']=='Southern Asia')]

# note: When using 'and' and 'or' operators, pandas requires we use '&' and '|' instead
of 'and' and 'or'
# don't forget to enclose the two conditions in parentheses
```

Out[37]:

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

9 rows × 38 columns

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

In [38]:

```
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', '19
92',
       '1993', '1994', '1995', '1996', '1997', '1998', '1999', '2000', '20
01',
       '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '20
10',
       '2011', '2012', '2013', 'Total'],
      dtype='object')
```

Out[38]:

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

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/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01). 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 [39]:

```
# we are using the inline backend
%matplotlib inline

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

*optional: check if Matplotlib is loaded.

In [40]:

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

```
Matplotlib version:  3.5.0
```

*optional: apply a style to Matplotlib.

In [41]:

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

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-n
ogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'g
gplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 's
eaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep',
'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 's
eaborn-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 (http://pandas.pydata.org/pandas-docs/stable/api.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utrSkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01#plotting)
- Plotting with Dataframes (http://pandas.pydata.org/pandas-docs/stable/api.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utrSkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01#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 [42]:

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

Out[42]:

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

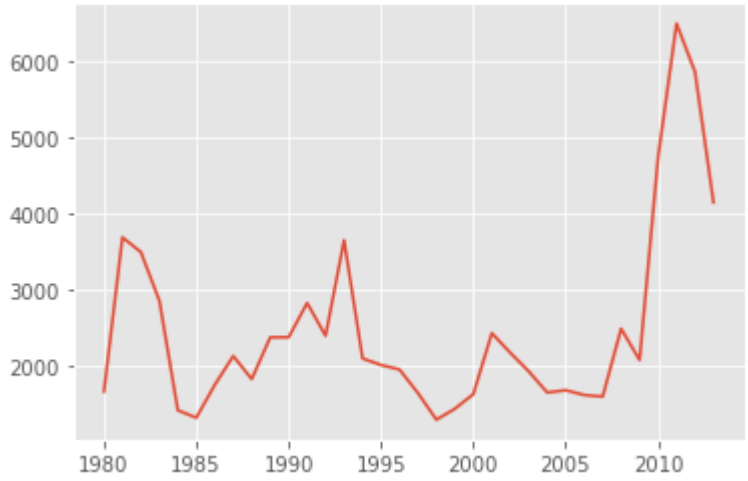
Next, we will plot a line plot by appending `.plot()` to the `haiti` dataframe.

In [43]:

```
haiti.plot()
```

Out[43]:

```
<AxesSubplot:>
```

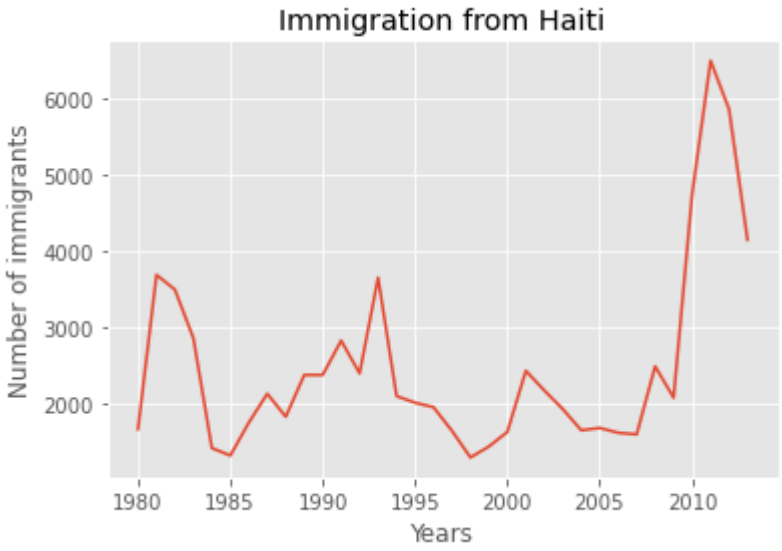



*pandas* automatically populated the x-axis with the index values (years), and the y-axis with the column values (population). However, notice how the years were not displayed because they are of type *string*. Therefore, let's change the type of the index values to *integer* for plotting.

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

In [44]:

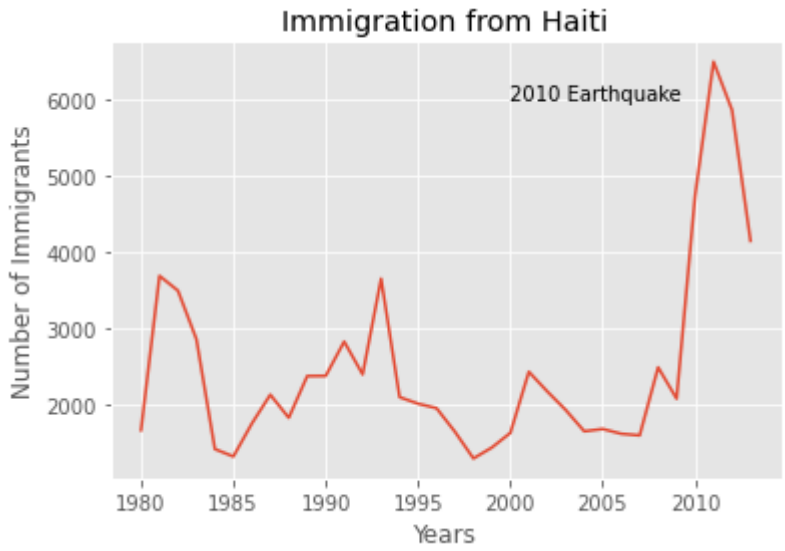
```
haiti.index = haiti.index.map(int) # let's change the index values of Haiti to type int
eger for plotting
haiti.plot(kind='line')

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

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




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

In [45]:

```
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 axi
s (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 ind
ex position of the year. Eg 20th index is year 2000 since it is the 20th year wi
th 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.

In [46]:

```
### type your answer here
df_CI = df_can.loc[['India', 'China'], years]
df_CI
```

Out[46]:

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

2 rows × 34 columns

Click here for a sample python solution

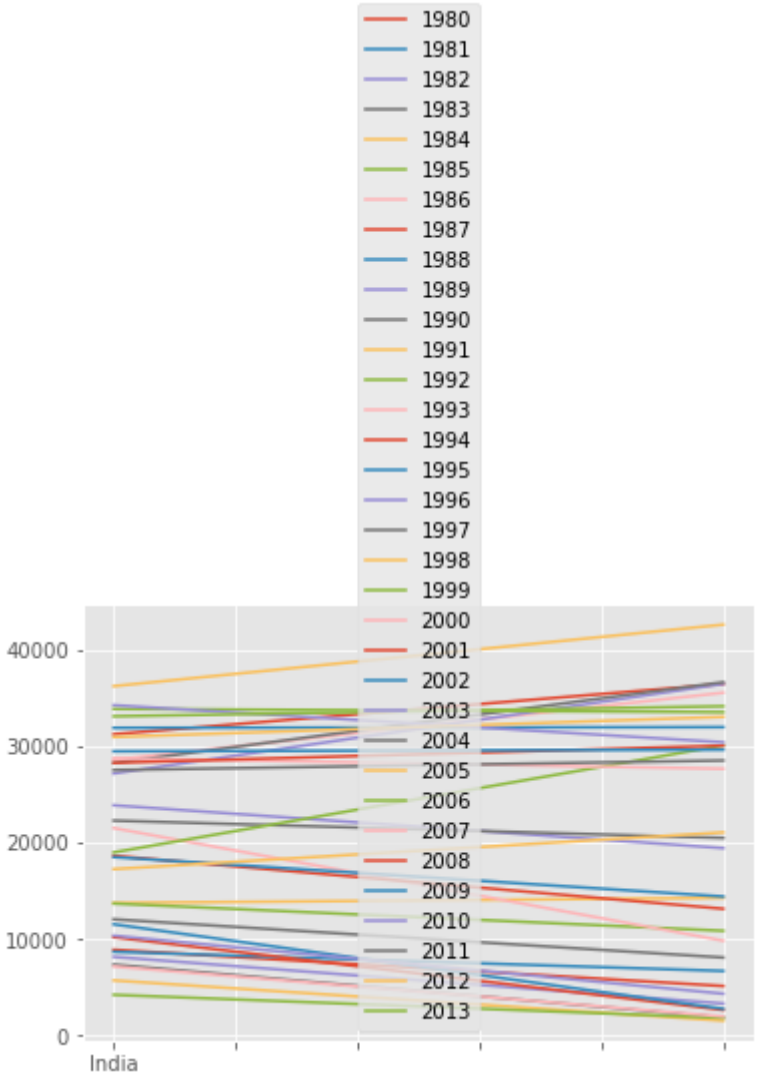
Step 2: Plot graph. We will explicitly specify line plot by passing in `kind` parameter to `plot()` .

In [47]:

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

Out[47]:

<AxesSubplot:>




Click here for a sample python solution

That doesn't look right...

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

In [48]:

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

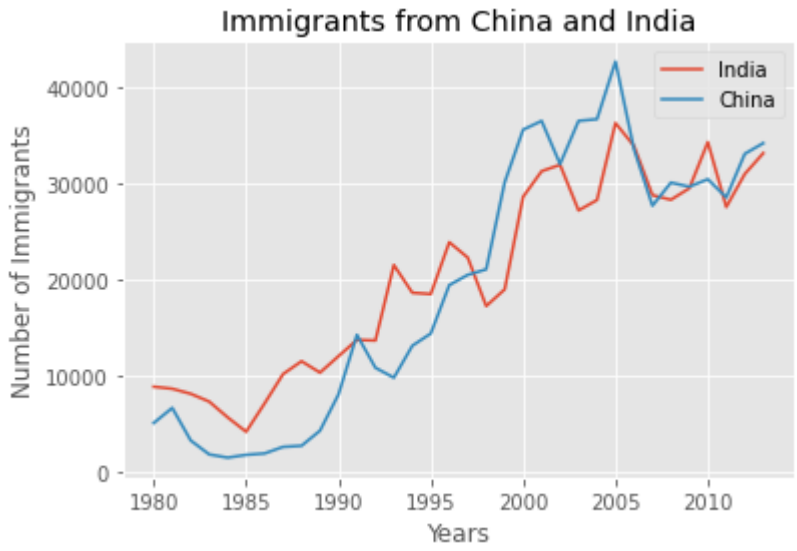
Out[48]:

| | 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 [49]:

```
### type your answer here
df_CI.index = df_CI.index.map(int) # let's change the index values of df_CI to type int
eger for plotting
df_CI.plot(kind='line')

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

plt.show()
```




Click here for a sample python solution

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

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

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

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

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

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

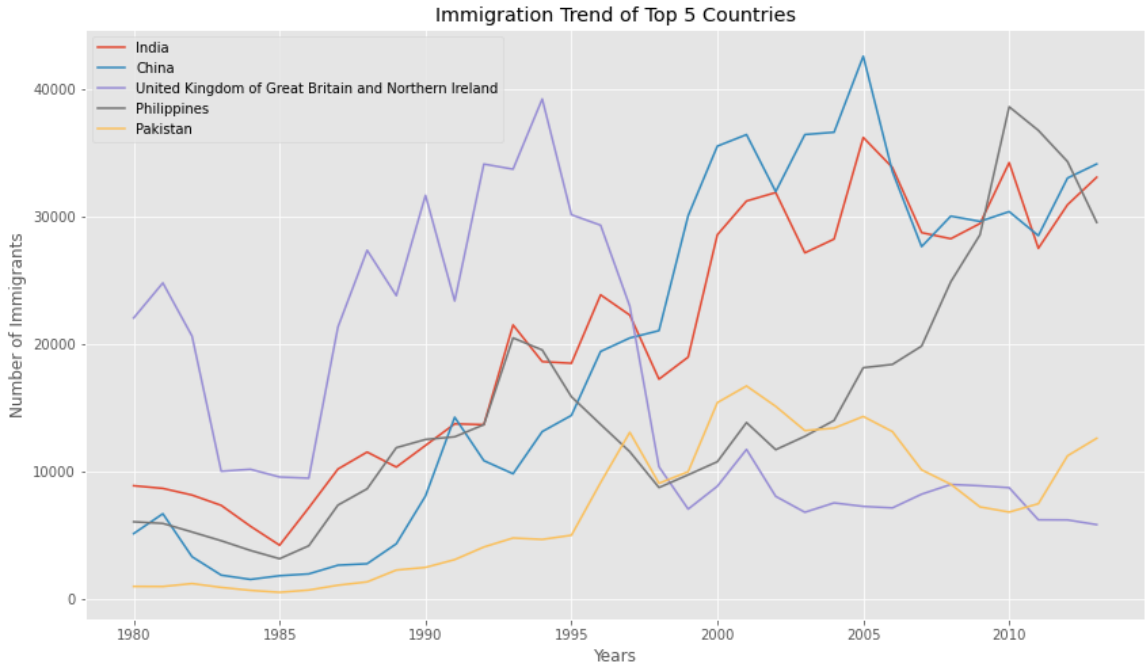
**Question:** Compare the trend of top 5 countries that contributed the most to immigration to Canada.

In [50]:

```
### type your answer here
#The correct answer is:
#Step 1: Get the dataset. Recall that we created a Total column that calculates cumulat
ive immigration by country.
#We will sort on this column to get our top 5 countries using pandas sort_values() meth
od.
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 u
sing the `figsize` parameter.
df_top5.index = df_top5.index.map(int) # let's change the index values of df_top5 to ty
pe integer for plotting
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()
```

| | India | China | United Kingdom of Great Britain and Northern Ireland | Philippines | Pakistan |
|---|---|---|---|---|---|
| 1980 | 8880 | 5123 | 22045 | 6051 | 978 |
| 1981 | 8670 | 6682 | 24796 | 5921 | 972 |
| 1982 | 8147 | 3308 | 20620 | 5249 | 1201 |
| 1983 | 7338 | 1863 | 10015 | 4562 | 900 |
| 1984 | 5704 | 1527 | 10170 | 3801 | 668 |
| 1985 | 4211 | 1816 | 9564 | 3150 | 514 |
| 1986 | 7150 | 1960 | 9470 | 4166 | 691 |
| 1987 | 10189 | 2643 | 21337 | 7360 | 1072 |
| 1988 | 11522 | 2758 | 27359 | 8639 | 1334 |
| 1989 | 10343 | 4323 | 23795 | 11865 | 2261 |
| 1990 | 12041 | 8076 | 31668 | 12509 | 2470 |
| 1991 | 13734 | 14255 | 23380 | 12718 | 3079 |
| 1992 | 13673 | 10846 | 34123 | 13670 | 4071 |
| 1993 | 21496 | 9817 | 33720 | 20479 | 4777 |
| 1994 | 18620 | 13128 | 39231 | 19532 | 4666 |
| 1995 | 18489 | 14398 | 30145 | 15864 | 4994 |
| 1996 | 23859 | 19415 | 29322 | 13692 | 9125 |
| 1997 | 22268 | 20475 | 22965 | 11549 | 13073 |
| 1998 | 17241 | 21049 | 10367 | 8735 | 9068 |
| 1999 | 18974 | 30069 | 7045 | 9734 | 9979 |
| 2000 | 28572 | 35529 | 8840 | 10763 | 15400 |
| 2001 | 31223 | 36434 | 11728 | | |
| 2002 | 31889 | 31961 | 8046 | | |
| 2003 | 27155 | 36439 | 6797 | | |
| 2004 | 28235 | 36619 | 7533 | | |
| 2005 | 36210 | 42584 | 7258 | | |
| 2006 | 33848 | 33518 | 7140 | | |
| 2007 | 28742 | 27642 | 8216 | | |
| 2008 | 28261 | 30037 | 8979 | | |
| 2009 | 29456 | 29622 | 8876 | | |
| 2010 | 34235 | 30391 | 8724 | | |
| 2011 | 27509 | 28502 | 6204 | | |
| 2012 | 30933 | 33024 | 6195 | | |
| 2013 | 33087 | 34129 | 5827 | | |

| | | |
|---|---|---|
| 2001 | 13836 | 16708 |
| 2002 | 11707 | 15110 |
| 2003 | 12758 | 13205 |
| 2004 | 14004 | 13399 |
| 2005 | 18139 | 14314 |
| 2006 | 18400 | 13127 |
| 2007 | 19837 | 10124 |
| 2008 | 24887 | 8994 |
| 2009 | 28573 | 7217 |
| 2010 | 38617 | 6811 |
| 2011 | 36765 | 7468 |
| 2012 | 34315 | 11227 |
| 2013 | 29544 | 12603 |




Click here for a sample python solution

## Other Plots

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

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

### Thank you for completing this lab!

## Author

Alex Aklson (https://www.linkedin.com/in/aklson/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01)

### Other Contributors

Jay Rajasekharan (https://www.linkedin.com/in/jayrajasekharan?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01), Ehsan M. Kermani (https://www.linkedin.com/in/ehsanmkermani?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01), Slobodan Markovic (https://www.linkedin.com/in/slobodan-markovic?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01), Weiqing Wang (https://www.linkedin.com/in/weiqing-wang-641640133/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01)

## Change Log

| Date (YYYY-MM-DD) | Version | Changed By | Change Description |
|---|---|---|---|
| 2021-05-29 | 2.4 | Weiqing Wang | Fixed typos and code smells. |
| 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 |