

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 <u>Python</u> <u>Basics for Data Science</u> and <u>Analyzing Data with Python</u>.

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**.

Table of Contents

- 1. Exploring Datasets with pandas
- 2. 1 The Dataset: Immigration to Canada from 1980 to 2013
- 3. 2 pandas Basics
- 4. 3 pandas Intermediate: Indexing and Selection
- 5. Visualizing Data using Matplotlib
- 6. 1 Matplotlib: Standard Python Visualization Library

7. Line Plots

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 familiarize yourself with the *pandas* API Reference: http://pandas.pydata.org/pandas-docs/stable/api.html.

The Dataset: Immigration to Canada from 1980 to 2013

Dataset Source: International migration flows to and from selected countries - The 2015 revision.

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.

Australe Residence Imagents Clicered	Sugg	rnational Migration Flow December 2015 - Copgested citation: United Nations, D	United Nations Population Division int of Economic and Social Affair is to and from Selected Countrie POP/DBMIGFlow/Rev.2015 yright © 2015 by United Nations. All right perpartment of Economic and Social Affairs, countries: The 2015 Revision, (United Nation	s: The 2015 Revision ghts reserved Population Division (2015).	15).																	
Residence	rting country																					
America Residence Immigrants Dech 100					▼ 1980 ▼	1981 💌	1982 💌	1983 💌	1984 💌	1985 💌	1986 💌	1987 💌	1988 💌	1989 💌	1990 💌	1991 💌	1992 💌	1993 💌	1994 💌	1995 💌	1996 💌	1997 💌
Australa Relationce Empirem Both 9950 8500 9250 19500																						
Australia Readence Emigrants Citeres																						
Austria Citizenship Emgrates Emgrate																	143660			149360	158260	176560
Austria Citerenin Citere					184290	212690	195200	153570	153530	172550	196690	221620	253860	238050	234050	237240	220460	197940	221920	253940	261330	260220
Austria Cligorathip Imrigraris Cligorath Cligorathip Cligorath																					17136	18830
Austria Citizensin Imrigrants Citizens			Emigrants																		46725	48264
Austria Residence inergarists 600			Immigrants																		12830	13227
Auchain Residence Engrands 60m		Citizenship	Immigrants	Foreigners																	50035	49638
Azerbaljan Residence Engrigoris Bom	la	Residence	Emigrants	Both																		
Acerbalan Residence Infogrants Both	ia	Residence	Immigrants																		69930	70122
Selection Residence Emjorate Selection Selec	saijan	Residence	Emigrants																	16033	13151	15703
Selection Residence Emjorate Both Selection	salian	Residence	Immigrants	Both																6222	5781	7528
Belguim Citemenip Engrants Citemen 1320 2025 2449 2100 2025 2244 1210 2225 1234 1200 1297 1802 1235 1341 1402 1442 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1462 1460 1	us			Both																		
Begium Citemonis Emgrants Foreignes 39887 39670 3770 39170 32747 39451 28660 31671 28667 28737 31471 24667 28741 32462 37750 28750 2	us	Residence	Immigrants	Both																		
Begium Citemenho Emgrants Foreignens 59887 59870 3770 3770 3770 32471 32467 22473 31471 24687 2447 32473 31471 24687 24472 32473 31471 24687 24472 32473 31471 24687 24472 32473 31471 24687 24472 32473 31471 24687 24472 32473 31471 24687 24472 32473 31471 24687 24472 34473 31471 24687 24472 34473 31471 24687 24472 34473 31471 24687 24472 34473 34471 34472 34473 34472 34473 34472 34473 34472 34473 34472 34473 34472 34473 34472 34473 34472 34473 34472 34473 34472 34473 34472 34473 34473 34472 34473 34474 34473 34472 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 34473 34474 344	um	Citizenship	Emigrants	Citizens	13326	20325	21497	21090	20562	20481	21110	22253	16244	16076	15937	18002	13258	13616	14422	16442	16384	18250
Beglum Cilzemshe Imrigrants Cilzemshe Cilzems	um	Citizenship		Foreigners	36887	36970	37207	36170	32747	30431	29509	31017	28981	24737	24373	31617	24597	29412	32462	31745	30616	32710
Seguin		Citizenship			7834	7979	8479	9310	9843	9500	9663	9655	10253	10620	12193	13330	11713	10707	10182	9812	9638	9609
Beginn Residence Imrigrants Dion S4894 49298 44959 47002 47042 48999 48795 48795 48845 54109 68793 68749		Citizenship		Foreigners	39746	33907	29498	28477	29884	28809	29466	31468	31343	35084	39338	41783	43312	48344	51034	45614	47716	45067
Sugaria Citemonish Emgratia Citemonish Emgratia Constitution Emgratia	um	Residence	Emigrants	Both																		-
Sugaria Citembule Emjorate Citembule Emjorate Citembule Emjorate Citembule Emjorate Citembule Emjorate Consignation Citembule Emjorate Citembule	um	Residence	Immigrants	Both	54694	49298	44659	43657	47002	47042	48959	49750	48484	54169	62662	67460	66763	63749				
Sugarian Citemonius Emgranta Emgranta Citemonius Emgranta Citemonius	ria	Citizenship		Citizens																		
Sugaria Citezenius Imrigurata Citezenius Imrigurata Citezenius Imrigurata Consensus Imrigurata Consensus Imrigurata Consensus Imrigurata Consensus Imrigurata Consensus Imrigurata Citezenius	ria																				- "	
Sugaria Citemonis Imrigants Foreigners	ıria																					
Sugarian Residence Emgrants Sub. S																						
Sugarian Residence Imrigrants Subri																						
Caresta Citezenship Immigraris Citezenship Citez																						
Canada Citemenip Immigrants Foreignes 14313 128641 12175 89185 88272 84346 99351 152707 161985 19195 216448 22279 247483 256835 224887 222887															3	3	4	3	- 1	- 1	- 1	
Contain					143137	128641	121175	89185	88272	84346	99351	152075	161585	191550	216448	232799	254783	256635	224381	212863	226070	216036
Coration Citizenship Engrands Engran								00.00	002.12	01010		102010	101000	101000	210110	EGE! GG	201100	200000	EE 1001	E-E-000	ELCC. C	E10000
Coration Citizenship Immigrants Citizenship Immigrants Citizenship Immigrants Coration																						
Cotation Citizenship Immigrates Foreigners																						
Cotation							-									- "		- "				
Costain																-	8859	9169	10163	15413	10027	18531
Open							- "									10050				42026	44596	52343
Open																10000	40024	57702	55-20	42020	44030	02.040
Öprind Oltzennialp Immigrants Oltzennia -																						
Öyrina Ölksensinip Immigrands Foroigners —																						
Oproval Residence Emigrands Both																						
Cyrous				Roth			- "														- "	
Cache Republic Citizenship Emigrands Citizens																	9904					6149
Cache Republic Citizenship Emigrants Foniquens																	0004					0140
Cacch Republic Citizenship Immigrants Citizens	h Dopublic																					
Cache Republic Collecterolity Immigrants Foreigners																						
Clasch Republic Residence Emigrants Both -																						
Casch Republic Residence Immigrants Both 10207 10540 Demmark Observable Observable 17979 18650 17991 16869 11962 18666 19881 23838 25447 23528 22167 22557 22557 22557 22350 22819 23252 Demmark Observable Emigrants Foreigners 11845 11077 10014 9122 8305 9171 9375 10068 10455 9273 8845 10185 9981 9614 101991 11199			Emigranto		-													7410	204	540	728	804
Dermank Citzonship Engants Citzonship Engants Citzonship Engants (1886) 17991 18850 17991 18850 17991 18850 17992 1893 2542 2352 2252 2252 2252 2359 2325 2591 2325 25																		7410			10857	12880
Denmark Citizenship Emigrants Foreigners 11845 11077 10014 9122 8305 9171 9375 10066 10455 9273 8845 10185 9081 9814 10891 11199					47070	10050	17001	10010	46800	47000	10000	10001	22002	25447	22520	22467	22557	22250			24355	12880 24336
																					24355 12809	24336 14033
				Foreigners																		
			Immigrants		14526							16239						∠2921			22918	22694
Denmark Cilizenship Immigrants Foreigners 15282 12982 12606 11433 12900 19219 20052 18217 16756 16996 17739 19744 19539 19623 20469 38231	nark	Citizenship	Immigrants	Foreigners	15282	12982	12606	11433	12900	19219	20052	18217	16756	16996	17739	19744	19539	19623	20469	38238	28914	2695

The Canada Immigration dataset can be fetched from here.

pandas Basics

The first thing we'll do is install **openpyxl** (formerly **xlrd**), a module that *pandas* requires to read Excel files.

```
Usage:
    pip3 install [options] <requirement specifier> [package-index-options] ...
    pip3 install [options] -r <requirements file> [package-index-options] ...
    pip3 install [options] [-e] <vcs project url> ...
    pip3 install [options] [-e] <local project path> ...
    pip3 install [options] <archive url/path> ...
    no such option: -y
```

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_excel() method.

```
df_can = pd.read_excel(
    'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevelo
    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.he

→		Туре	Coverage	0dName	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.

df_can.tail()



	Туре	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.

df_can.info(verbose=False)

<<rp><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.

df_can.columns

→	<pre>Index('REG',</pre>	- //	'Coverage',	'OdName',	'AREA',	'AreaName',
	1982,	'RegName',	'DEV',	'DevName',	1980,	1981,
	1982,	1983,	1984,	1985,	1986,	1987,
	1994,	1989,	1990,	1991,	1992,	1993,
	2000,	1995,	1996,	1997,	1998,	1999,
		2001,	2002,	2003,	2004,	2005,
	2006,	2007,	2008,	2009,	2010,	2011,
	2012,	2013] dtype='object				

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

```
df_can.index

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

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

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

```
df_can.columns.tolist()
     ['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]
```

```
df_can.index.tolist()
     [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,

```
24/06/2024, 16:58
```

```
49,
50,
51,
52,
53,
54,
55,
56,
57.
```

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

To view the dimensions of the dataframe, we use the shape instance variable of it.

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

(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 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)
```

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

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:

df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'RegName':'Reg
df_can.columns

```
→ Index([ 'Country', 'Continent', 'Region', 'DevName', 1980, 1981, 1982, 1983, 1984, 1985,
```

```
1986,
                       1987,
                                     1988,
                                                   1989.
                                                                 1990,
                       1992,
        1991,
                                     1993,
                                                   1994,
                                                                 1995,
        1996,
                      1997,
                                     1998,
                                                   1999,
                                                                 2000,
                                     2003,
        2001,
                       2002,
                                                   2004,
                                                                 2005,
                                                   2009,
        2006,
                                     2008,
                      2007,
                                                                 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:

```
Start coding or generate with AI.
```

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

df_can.isnull().sum()

\rightarrow	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 dtype: int64

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

df_can.describe()

6		_
÷	۸	÷
	_	ı.
	_	_

	1980	1981	1982	1983	1984	198
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.00000
mean	508.394872	566.989744	534.723077	387.435897	376.497436	358.86153
std	1949.588546	2152.643752	1866.997511	1204.333597	1198.246371	1079.30960
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
50%	13.000000	10.000000	11.000000	12.000000	13.000000	17.00000
75%	251.500000	295.500000	275.000000	173.000000	181.000000	197.00000
max	22045.000000	24796.000000	20620.000000	10015.000000	10170.000000	9564.00000

8 rows × 34 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').

df_can.Country # returns a series

→	0 1 2 3	America				
	4	,	Andorra			
			•			
	190	V	iet Nam			
	191	Western	Sahara			
	192		Yemen			
	193		Zambia			
	194	Z	imbabwe			
	Name:	Country,	Length:	195,	dtype:	object

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

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 o

→		Country	1980	1981	1982	1983	1984	1985	
	0	Afghanistan	16	39	39	47	71	340	ılı
	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.

```
df_can.set_index('Country', inplace=True)
# tip: The opposite of set is reset. So to reset the index, we can use df_can.res
df can.head(3)
```

→		Continent	Region	DevName	1980	1981	1982	1983	1984	1985	19
	Country										
	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	4
	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	
	Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	

3 rows × 37 columns

```
# 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

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

$\overline{\Rightarrow}$	Continent	Asia
	Region	Eastern Asia
	DevName	Developed regions
	1980	701
	1981	756
	1982	598
	1983	309
	1984	246
	1985	198
	1986	248
	1987	422

30			DVUIUIEN-EXER
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
Name:	Japan,	dtype:	object

Name: Japan, dtype: object

alternate methods df_can.iloc[87]

\rightarrow	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
                             1212
2006
                             1250
2007
                             1284
2008
2009
                             1194
2010
                             1168
2011
                             1265
2012
                             1214
2013
                              982
```

Name: Japan, dtype: object

df_can[df_can.index == 'Japan']

→		Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	• • •
	Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198	248	

1 rows × 37 columns

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

→ 982

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

→ 982

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

1980 701 1981 756 1982 598 1983 309 1984 246 1984 246

Name: Japan, dtype: object

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

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'.

```
df_can.columns = list(map(str, df_can.columns))
# [print (type(x)) for x in df_can.columns.values] #<-- uncomment to check type o</pre>
```

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']
```

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).

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

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

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



	Continent	Region	DevName	1980	1981	1982	1983	1984	1985
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340
Armenia	Asia	Western Asia	Developing regions	0	0	0	0	0	0
Azerbaijan	Asia	Western Asia	Developing regions	0	0	0	0	0	0
Bahrain	Asia	Western Asia	Developing regions	0	2	1	1	1	3
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0
Brunei Darussalam	Asia	South- Eastern Asia	Developing regions	79	6	8	2	2	4
Cambodia	Asia	South- Eastern Asia	Developing regions	12	19	26	33	10	7
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816
China, Hong Kong Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0
China, Macao Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0
Cyprus	Asia	Western Asia	Developing regions	132	128	84	46	46	43
Democratic People's Republic of Korea	Asia	Eastern Asia	Developing regions	1	1	3	1	4	3
Georgia	Asia	Western Asia	Developing regions	0	0	0	0	0	0
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211
Indonesia	Asia	South- Eastern Asia	Developing regions	186	178	252	115	123	100
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648
Iraa search google com/drive/1i	Acia SEMNy Odk NTS char	Western	Developing	つらつ iPnCnz04	OAF Juha V kuri	OAN atMada_tm	2 20	ΛOQ	921

8	DV0	101EN-Exercise Asia	-Introduction-to-Ma regions	tplotlib-and	d-Line-Plot	s (1).ipynb	- Colab	740	١٠٠
Israel	Asia	Western Asia	Developing regions	1403	1711	1334	541	446	680
Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198
Jordan	Asia	Western Asia	Developing regions	177	160	155	113	102	179
Kazakhstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0
Kuwait	Asia	Western Asia	Developing regions	1	0	8	2	1	4
Kyrgyzstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0
Lao People's Democratic Republic	Asia	South- Eastern Asia	Developing regions	11	6	16	16	7	17
Lebanon	Asia	Western Asia	Developing regions	1409	1119	1159	789	1253	1683
Malaysia	Asia	South- Eastern Asia	Developing regions	786	816	813	448	384	374
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0
Mongolia	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0
Myanmar	Asia	South- Eastern Asia	Developing regions	80	62	46	31	41	23
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4
Oman	Asia	Western Asia	Developing regions	0	0	0	8	0	0
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514
Philippines	Asia	South- Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150
Qatar	Asia	Western Asia	Developing regions	0	0	0	0	0	0
Republic of Korea	Asia	Eastern Asia	Developing regions	1011	1456	1572	1081	847	962
Saudi Arabia	Asia	Western Asia	Developing regions	0	0	1	4	1	2
<u> </u>		South-	Developing	244	221				

24/06/2024, 16:58 Singapore	DV0 ASI a	101EN-Exercise ⊏astern Asia	-Introduction-to-Ma · - regions	tplotlib-and	-Line-Plots 3 U1	(1).ipynb - 33 /	Colab 1 09	128	139
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845
State of Palestine	Asia	Western Asia	Developing regions	0	0	0	0	0	0
Syrian Arab Republic	Asia	Western Asia	Developing regions	315	419	409	269	264	385
Tajikistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0
Thailand	Asia	South- Eastern Asia	Developing regions	56	53	113	65	82	66
Turkey	Asia	Western Asia	Developing regions	481	874	706	280	338	202
Turkmenistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0
United Arab Emirates	Asia	Western Asia	Developing regions	0	2	2	1	2	0
Uzbekistan	Asia	Central	Developing	0	0	0	0	0	0

0

1191

1

regions

regions

regions

Developing

Developing

0

1829

0

2162

0

3404

0

7583

0

0

5907

18

49 rows × 37 columns

Uzbekistan

Viet Nam

Yemen

Asia

South-

Eastern

Western

Asia

Asia

Asia

Asia

Asia

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

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

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



	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	19
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	4
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	4
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	71
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	17
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	6
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	18

9 rows × 37 columns

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

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

Afghanistan Asia 16 39 39 47 71 340 Asia regions Southern Developed **Albania** Europe 1 0 0 0 0 Europe regions

2 rows × 37 columns

Visualizing Data using Matplotlib

Matplotlib: Standard Python Visualization Library

The primary plotting library we will explore in the course is <u>Matplotlib</u>. 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: 3.7.1
```

*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-nogri
```

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 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.

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

```
1980 1666
1981 3692
1982 3498
1983 2860
```

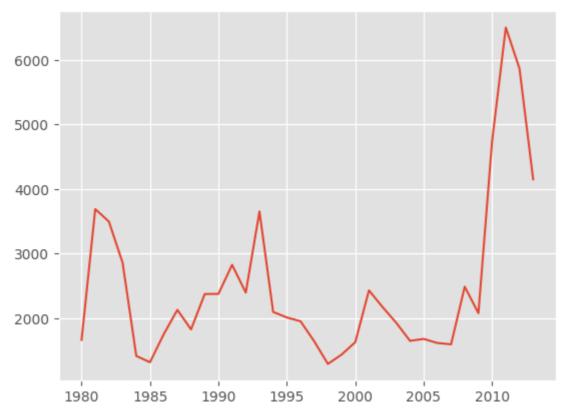
1984 1418

Name: Haiti, dtype: object

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

haiti.plot()

→ <Axes: >



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:

haiti.index = haiti.index.map(int) # let's change the index values of Haiti to ty 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

 $\overline{2}$

6000 -

5000 -

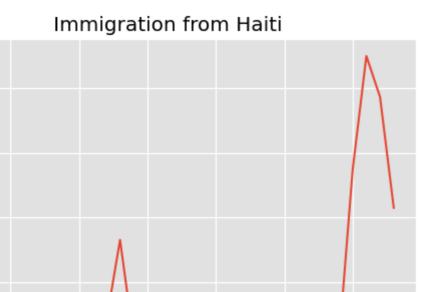
4000 -

3000 -

2000

1980

Number of immigrants



2000

2005

2010

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.

1995

Years

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

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

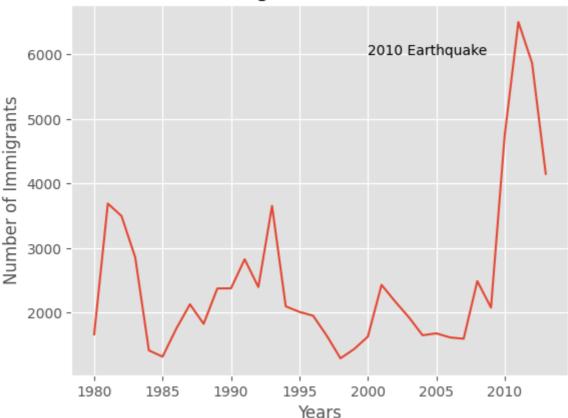
# annotate the 2010 Earthquake.
# syntax: plt.text(x, y, label)
plt.text(2000, 6000, '2010 Earthquake') # see note below
plt.show()
```

1985

1990

 \overline{z}





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 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

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.

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

```
\rightarrow
            1980 1981 1982 1983 1984
                                          1985 1986
                                                       1987
                                                              1988
                                                                    1989
                                                                                2004
     India
            8880
                  8670
                        8147
                              7338
                                    5704
                                           4211
                                                 7150 10189
                                                             11522
                                                                    10343
                                                                                28235
     China 5123
                  6682
                             1863 1527
                                                1960
                                                                                36619 4
                        3308
                                          1816
                                                       2643
                                                              2758
                                                                     4323
    2 rows × 34 columns
```

▼ Click here for a sample python solution

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

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

```
### type your answer here
# df_CI. ...
```

▼ 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.

```
df_CI = df_can.loc[['India', 'China'], years]
df_CI = df_CI.transpose()
df_CI.head()
```

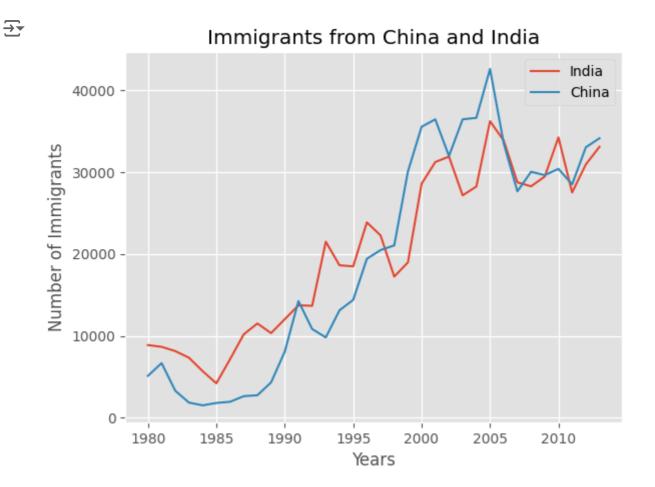


pandas will automatically graph the two countries on the same graph. Go ahead and plot the new transposed dataframe. Make sure to add a title to the plot and label the axes.

```
### type your answer here

df_CI.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

```
#The correct answer is:
     df_CI.index = df_CI.index.map(int) # let's change the index values of df_CI to type i
     df_CI.plot(kind='line')
     plt.title('Immigrants from China and India')
     plt.ylabel('Number of Immigrants')
     plt.xlabel('Years')
     plt.show()
df_CI.index = df_CI.index.map(int) # let's change the index values of df_CI to ty
df_CI.plot(kind='line')
plt.title('Immigrants from China and India')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
\rightarrow
```





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