immigrations

April 20, 2020

[43]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
!curl https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/
 ↪CognitiveClass/DV0101EN/labs/Data_Files/Canada.xlsx -o Canada.xlsx
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  381k  100  381k    0     0   702k      0 --:--:-- --:--:-- --:--:--  711k
```

[3]:
```
df = pd.read_excel('Canada.xlsx')
```

[4]:
```
df.head()
```

[4]:
```
                                  Unnamed: 0 Unnamed: 1 Unnamed: 2 Unnamed: 3  \
0                                        NaN        NaN        NaN        NaN
1                                        NaN        NaN        NaN        NaN
2                            United Nations        NaN        NaN        NaN
3                       Population Division        NaN        NaN        NaN
4  Department of Economic and Social Affairs        NaN        NaN        NaN

  Unnamed: 4 Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9  \
0        NaN        NaN        NaN        NaN        NaN        NaN
1        NaN        NaN        NaN        NaN        NaN        NaN
2        NaN        NaN        NaN        NaN        NaN        NaN
3        NaN        NaN        NaN        NaN        NaN        NaN
4        NaN        NaN        NaN        NaN        NaN        NaN

      ...      Unnamed: 28 Unnamed: 29 Unnamed: 30 Unnamed: 31 Unnamed: 32  \
0     ...              NaN         NaN         NaN         NaN         NaN
1     ...              NaN         NaN         NaN         NaN         NaN
2     ...              NaN         NaN         NaN         NaN         NaN
3     ...              NaN         NaN         NaN         NaN         NaN
4     ...              NaN         NaN         NaN         NaN         NaN

  Unnamed: 33 Unnamed: 34  Unnamed: 35 Unnamed: 36  Unnamed: 37
0         NaN         NaN          NaN         NaN          NaN
```

```
1         NaN         NaN         NaN         NaN         NaN
2         NaN         NaN         NaN         NaN         NaN
3         NaN         NaN         NaN         NaN         NaN
4         NaN         NaN         NaN         NaN         NaN

[5 rows x 38 columns]
```

[5]:
```
df_can = pd.read_excel('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!
```

[6]:
```
#Let's view the top 5 rows of the dataset using the head() function.
df_can.head()
```

[6]:

	Type	Coverage	OdName	AREA	AreaName	REG	\
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	
1	Immigrants	Foreigners	Albania	908	Europe	925	
2	Immigrants	Foreigners	Algeria	903	Africa	912	
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	
4	Immigrants	Foreigners	Andorra	908	Europe	925	

	RegName	DEV	DevName	1980	...	2004	2005	2006	\
0	Southern Asia	902	Developing regions	16	...	2978	3436	3009	
1	Southern Europe	901	Developed regions	1	...	1450	1223	856	
2	Northern Africa	902	Developing regions	80	...	3616	3626	4807	
3	Polynesia	902	Developing regions	0	...	0	0	1	
4	Southern Europe	901	Developed regions	0	...	0	0	1	

	2007	2008	2009	2010	2011	2012	2013
0	2652	2111	1746	1758	2203	2635	2004
1	702	560	716	561	539	620	603
2	3623	4005	5393	4752	4325	3774	4331
3	0	0	0	0	0	0	0
4	1	0	0	0	0	1	1

[5 rows x 43 columns]

[7]:
```
#We can also veiw the bottom 5 rows of the dataset using the tail() function.
df_can.tail()
```

[7]:

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

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

     2006  2007  2008  2009  2010  2011  2012  2013
190  3153  2574  1784  2171  1942  1723  1731  2112
191     1     0     0     0     0     0     0     0
192   140   122   133   128   211   160   174   217
193    77    71    64    60   102    69    46    59
194   454   663   611   508   494   434   437   407

[5 rows x 43 columns]
```

[8]:
```
#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.
df_can.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 43 columns):
Type        195 non-null object
Coverage    195 non-null object
OdName      195 non-null object
AREA        195 non-null int64
AreaName    195 non-null object
REG         195 non-null int64
RegName     195 non-null object
DEV         195 non-null int64
DevName     195 non-null object
1980        195 non-null int64
1981        195 non-null int64
1982        195 non-null int64
1983        195 non-null int64
1984        195 non-null int64
1985        195 non-null int64
1986        195 non-null int64
1987        195 non-null int64
1988        195 non-null int64
1989        195 non-null int64
1990        195 non-null int64
1991        195 non-null int64
1992        195 non-null int64
1993        195 non-null int64
1994        195 non-null int64
1995        195 non-null int64
```

```
1996        195 non-null int64
1997        195 non-null int64
1998        195 non-null int64
1999        195 non-null int64
2000        195 non-null int64
2001        195 non-null int64
2002        195 non-null int64
2003        195 non-null int64
2004        195 non-null int64
2005        195 non-null int64
2006        195 non-null int64
2007        195 non-null int64
2008        195 non-null int64
2009        195 non-null int64
2010        195 non-null int64
2011        195 non-null int64
2012        195 non-null int64
2013        195 non-null int64
dtypes: int64(37), object(6)
memory usage: 65.6+ KB
```

[9]:
```
#To get the list of column headers we can call upon the dataframe's .columns␣
 ↪parameter.
df_can.columns.values
```

[9]:
```
array(['Type', 'Coverage', 'OdName', 'AREA', 'AreaName', 'REG', 'RegName',
       'DEV', 'DevName', 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987,
       1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998,
       1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009,
       2010, 2011, 2012, 2013], dtype=object)
```

[10]:
```
#Similarly, to get the list of indicies(ROWS-INDEX) we use the .index parameter.
df_can.index.values
```

[10]:
```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
       156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
       169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
       182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194])
```

[11]:
```
print(type(df_can.columns))
print(type(df_can.index))
```

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

[12]:
```
#To get the index and columns as lists, we can use the tolist() method.
df_can.columns.tolist()

#HERE IT is giving the columns name in list form by using "tolist()" method
#print (type(df_can.columns.tolist()))
#print (type(df_can.index.tolist()))
```

[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]
```

[13]:
```
##To get the index and columns as lists, we can use the tolist() method.
df_can.index.tolist()
```

[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]
```

[14]:
```
#To view the dimensions of the dataframe, we use the .shape parameter.
df_can.shape
```

[14]:
```
(195, 43)
```

[15]:
```
df_can.head(1)
```

[15]:
```
         Type    Coverage       OdName  AREA AreaName   REG        RegName  \
0  Immigrants  Foreigners  Afghanistan   935     Asia  5501  Southern Asia

   DEV             DevName  1980  ...   2004  2005  2006  2007  2008  2009  \
0  902  Developing regions    16  ...   2978  3436  3009  2652  2111  1746

   2010  2011  2012  2013
0  1758  2203  2635  2004

[1 rows x 43 columns]
```

[16]:
```
#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_dropping =df_can.drop(['AREA','REG','DEV','Type','Coverage'], axis=1,␣
 ↪inplace=False)
df_dropping.head(2)
```

[16]:
```
        OdName AreaName          RegName             DevName  1980  1981  \
0  Afghanistan     Asia    Southern Asia  Developing regions    16    39
1      Albania   Europe  Southern Europe   Developed regions     1     0

   1982  1983  1984  1985  ...   2004  2005  2006  2007  2008  2009  2010  \
0    39    47    71   340  ...   2978  3436  3009  2652  2111  1746  1758
```

```
1     0     0     0     0  ...   1450  1223   856   702   560   716   561

   2011  2012  2013
0  2203  2635  2004
1   539   620   603

[2 rows x 38 columns]
```

[17]:
```
#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_dropping.rename(columns={'OdName':'Country', 'AreaName':'Continent',␣
 ↪'RegName':'Region'}, inplace=True)
df_dropping.columns
```

[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')
```

[18]:
```
df_dropping.head()
```

[18]:
```
          Country Continent          Region            DevName  1980  1981  \
0     Afghanistan      Asia    Southern Asia  Developing regions    16    39
1         Albania    Europe  Southern Europe   Developed regions     1     0
2         Algeria    Africa  Northern Africa  Developing regions    80    67
3  American Samoa   Oceania       Polynesia  Developing regions     0     1
4         Andorra    Europe  Southern Europe   Developed regions     0     0

   1982  1983  1984  1985  ...   2004  2005  2006  2007  2008  2009  2010  \
0    39    47    71   340  ...   2978  3436  3009  2652  2111  1746  1758
1     0     0     0     0  ...   1450  1223   856   702   560   716   561
2    71    69    63    44  ...   3616  3626  4807  3623  4005  5393  4752
3     0     0     0     0  ...      0     0     1     0     0     0     0
4     0     0     0     0  ...      0     0     1     1     0     0     0

   2011  2012  2013
0  2203  2635  2004
1   539   620   603
2  4325  3774  4331
3     0     0     0
4     0     1     1

[5 rows x 38 columns]
```

[19]:
```
#We will also add a 'Total' column that sums up the total immigrants by country␣
 ↪over the entire period 1980 - 2013, as follows:
#The "sum() method only add up integer/float therefore it is adding only integer␣
 ↪not adding string(object) type"
df_dropping['Total'] = df_dropping.sum(axis=1)
```

[20]:
```
#We can check to see how many null objects we have in the dataset as follows,␣
 ↪here "False" mean that it is not empty
df_dropping.isnull()
```

[20]:

	Country	Continent	Region	DevName	1980	1981	1982	1983	1984	\
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	
5	False	False	False	False	False	False	False	False	False	
6	False	False	False	False	False	False	False	False	False	
7	False	False	False	False	False	False	False	False	False	
8	False	False	False	False	False	False	False	False	False	
9	False	False	False	False	False	False	False	False	False	
10	False	False	False	False	False	False	False	False	False	
11	False	False	False	False	False	False	False	False	False	
12	False	False	False	False	False	False	False	False	False	
13	False	False	False	False	False	False	False	False	False	
14	False	False	False	False	False	False	False	False	False	
15	False	False	False	False	False	False	False	False	False	
16	False	False	False	False	False	False	False	False	False	
17	False	False	False	False	False	False	False	False	False	
18	False	False	False	False	False	False	False	False	False	
19	False	False	False	False	False	False	False	False	False	
20	False	False	False	False	False	False	False	False	False	
21	False	False	False	False	False	False	False	False	False	
22	False	False	False	False	False	False	False	False	False	
23	False	False	False	False	False	False	False	False	False	
24	False	False	False	False	False	False	False	False	False	
25	False	False	False	False	False	False	False	False	False	
26	False	False	False	False	False	False	False	False	False	
27	False	False	False	False	False	False	False	False	False	
28	False	False	False	False	False	False	False	False	False	
29	False	False	False	False	False	False	False	False	False	
..	...	...	...	...	...	...	...	...	...	
165	False	False	False	False	False	False	False	False	False	
166	False	False	False	False	False	False	False	False	False	
167	False	False	False	False	False	False	False	False	False	
168	False	False	False	False	False	False	False	False	False	
169	False	False	False	False	False	False	False	False	False	
170	False	False	False	False	False	False	False	False	False	

```
171    False      False   False    False  False  False  False  False  False
172    False      False   False    False  False  False  False  False  False
173    False      False   False    False  False  False  False  False  False
174    False      False   False    False  False  False  False  False  False
175    False      False   False    False  False  False  False  False  False
176    False      False   False    False  False  False  False  False  False
177    False      False   False    False  False  False  False  False  False
178    False      False   False    False  False  False  False  False  False
179    False      False   False    False  False  False  False  False  False
180    False      False   False    False  False  False  False  False  False
181    False      False   False    False  False  False  False  False  False
182    False      False   False    False  False  False  False  False  False
183    False      False   False    False  False  False  False  False  False
184    False      False   False    False  False  False  False  False  False
185    False      False   False    False  False  False  False  False  False
186    False      False   False    False  False  False  False  False  False
187    False      False   False    False  False  False  False  False  False
188    False      False   False    False  False  False  False  False  False
189    False      False   False    False  False  False  False  False  False
190    False      False   False    False  False  False  False  False  False
191    False      False   False    False  False  False  False  False  False
192    False      False   False    False  False  False  False  False  False
193    False      False   False    False  False  False  False  False  False
194    False      False   False    False  False  False  False  False  False

      1985  ...    2005   2006   2007   2008   2009   2010   2011   2012  \
0    False  ...   False  False  False  False  False  False  False  False
1    False  ...   False  False  False  False  False  False  False  False
2    False  ...   False  False  False  False  False  False  False  False
3    False  ...   False  False  False  False  False  False  False  False
4    False  ...   False  False  False  False  False  False  False  False
5    False  ...   False  False  False  False  False  False  False  False
6    False  ...   False  False  False  False  False  False  False  False
7    False  ...   False  False  False  False  False  False  False  False
8    False  ...   False  False  False  False  False  False  False  False
9    False  ...   False  False  False  False  False  False  False  False
10   False  ...   False  False  False  False  False  False  False  False
11   False  ...   False  False  False  False  False  False  False  False
12   False  ...   False  False  False  False  False  False  False  False
13   False  ...   False  False  False  False  False  False  False  False
14   False  ...   False  False  False  False  False  False  False  False
15   False  ...   False  False  False  False  False  False  False  False
16   False  ...   False  False  False  False  False  False  False  False
17   False  ...   False  False  False  False  False  False  False  False
18   False  ...   False  False  False  False  False  False  False  False
19   False  ...   False  False  False  False  False  False  False  False
20   False  ...   False  False  False  False  False  False  False  False
```

```
21   False  ...    False  False  False  False  False  False  False  False
22   False  ...    False  False  False  False  False  False  False  False
23   False  ...    False  False  False  False  False  False  False  False
24   False  ...    False  False  False  False  False  False  False  False
25   False  ...    False  False  False  False  False  False  False  False
26   False  ...    False  False  False  False  False  False  False  False
27   False  ...    False  False  False  False  False  False  False  False
28   False  ...    False  False  False  False  False  False  False  False
29   False  ...    False  False  False  False  False  False  False  False
..     ...  ...      ...    ...    ...    ...    ...    ...    ...    ...
165  False  ...    False  False  False  False  False  False  False  False
166  False  ...    False  False  False  False  False  False  False  False
167  False  ...    False  False  False  False  False  False  False  False
168  False  ...    False  False  False  False  False  False  False  False
169  False  ...    False  False  False  False  False  False  False  False
170  False  ...    False  False  False  False  False  False  False  False
171  False  ...    False  False  False  False  False  False  False  False
172  False  ...    False  False  False  False  False  False  False  False
173  False  ...    False  False  False  False  False  False  False  False
174  False  ...    False  False  False  False  False  False  False  False
175  False  ...    False  False  False  False  False  False  False  False
176  False  ...    False  False  False  False  False  False  False  False
177  False  ...    False  False  False  False  False  False  False  False
178  False  ...    False  False  False  False  False  False  False  False
179  False  ...    False  False  False  False  False  False  False  False
180  False  ...    False  False  False  False  False  False  False  False
181  False  ...    False  False  False  False  False  False  False  False
182  False  ...    False  False  False  False  False  False  False  False
183  False  ...    False  False  False  False  False  False  False  False
184  False  ...    False  False  False  False  False  False  False  False
185  False  ...    False  False  False  False  False  False  False  False
186  False  ...    False  False  False  False  False  False  False  False
187  False  ...    False  False  False  False  False  False  False  False
188  False  ...    False  False  False  False  False  False  False  False
189  False  ...    False  False  False  False  False  False  False  False
190  False  ...    False  False  False  False  False  False  False  False
191  False  ...    False  False  False  False  False  False  False  False
192  False  ...    False  False  False  False  False  False  False  False
193  False  ...    False  False  False  False  False  False  False  False
194  False  ...    False  False  False  False  False  False  False  False

      2013  Total
0    False  False
1    False  False
2    False  False
3    False  False
4    False  False
```

```
5    False  False
6    False  False
7    False  False
8    False  False
9    False  False
10   False  False
11   False  False
12   False  False
13   False  False
14   False  False
15   False  False
16   False  False
17   False  False
18   False  False
19   False  False
20   False  False
21   False  False
22   False  False
23   False  False
24   False  False
25   False  False
26   False  False
27   False  False
28   False  False
29   False  False
..     ...    ...
165  False  False
166  False  False
167  False  False
168  False  False
169  False  False
170  False  False
171  False  False
172  False  False
173  False  False
174  False  False
175  False  False
176  False  False
177  False  False
178  False  False
179  False  False
180  False  False
181  False  False
182  False  False
183  False  False
184  False  False
185  False  False
```

```
186  False  False
187  False  False
188  False  False
189  False  False
190  False  False
191  False  False
192  False  False
193  False  False
194  False  False

[195 rows x 39 columns]
```

[28]:
```
#Here it will sum up all the BOX by column wise and if "True" occurs then it␣
 ↪will count +1.
df_dropping.isnull().sum()
```

[28]:
```
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
```

[21]:
```
#Finally, let's view a quick summary of each column in our dataframe using the␣
 ↪describe() method.
df_dropping.describe()
```

[21]:

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

	1985	1986	1987	1988	1989	\
count	195.000000	195.000000	195.000000	195.000000	195.000000	
mean	358.861538	441.271795	691.133333	714.389744	843.241026	
std	1079.309600	1225.576630	2109.205607	2443.606788	2555.048874	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.500000	0.500000	1.000000	1.000000	
50%	17.000000	18.000000	26.000000	34.000000	44.000000	
75%	197.000000	254.000000	434.000000	409.000000	508.500000	
max	9564.000000	9470.000000	21337.000000	27359.000000	23795.000000	

	...	2005	2006	2007	2008	\
count	...	195.000000	195.000000	195.000000	195.000000	
mean	...	1320.292308	1266.958974	1191.820513	1246.394872	
std	...	4425.957828	3926.717747	3443.542409	3694.573544	
min	...	0.000000	0.000000	0.000000	0.000000	
25%	...	28.500000	25.000000	31.000000	31.000000	
50%	...	210.000000	218.000000	198.000000	205.000000	
75%	...	832.000000	842.000000	899.000000	934.500000	
max	...	42584.000000	33848.000000	28742.000000	30037.000000	

	2009	2010	2011	2012	2013	\
count	195.000000	195.000000	195.000000	195.000000	195.000000	
mean	1275.733333	1420.287179	1262.533333	1313.958974	1320.702564	
std	3829.630424	4462.946328	4030.084313	4247.555161	4237.951988	
min	0.000000	0.000000	0.000000	0.000000	0.000000	

```
25%       36.000000     40.500000     37.500000     42.500000     45.000000
50%      214.000000    211.000000    179.000000    233.000000    213.000000
75%      888.000000    932.000000    772.000000    783.000000    796.000000
max    29622.000000  38617.000000  36765.000000  34315.000000  34129.000000

               Total
count     195.000000
mean    32867.451282
std     91785.498686
min         1.000000
25%       952.000000
50%      5018.000000
75%     22239.500000
max    691904.000000

[8 rows x 35 columns]
```

1 Select Column

There are two ways to filter on a column name:

Method 1: Quick and easy, but only works if the column name does NOT have spaces or special characters.

```
df.column_name
    (returns series)
```

Method 2: More robust, and can filter on multiple columns.

```
df['column']
    (returns series)
df[['column 1', 'column 2']]
    (returns dataframe)
```

[22]:
```
df_dropping.Country  # returns a series
```

[22]:
```
0                                        Afghanistan
1                                            Albania
2                                            Algeria
3                                     American Samoa
4                                            Andorra
5                                             Angola
6                                  Antigua and Barbuda
7                                          Argentina
8                                            Armenia
9                                          Australia
10                                           Austria
11                                        Azerbaijan
12                                           Bahamas
```

```
13                                          Bahrain
14                                       Bangladesh
15                                         Barbados
16                                          Belarus
17                                          Belgium
18                                           Belize
19                                            Benin
20                                           Bhutan
21                    Bolivia (Plurinational State of)
22                              Bosnia and Herzegovina
23                                         Botswana
24                                           Brazil
25                                  Brunei Darussalam
26                                         Bulgaria
27                                      Burkina Faso
28                                          Burundi
29                                       Cabo Verde
                            ...
165                                       Suriname
166                                      Swaziland
167                                         Sweden
168                                     Switzerland
169                               Syrian Arab Republic
170                                      Tajikistan
171                                       Thailand
172          The former Yugoslav Republic of Macedonia
173                                            Togo
174                                           Tonga
175                               Trinidad and Tobago
176                                         Tunisia
177                                          Turkey
178                                      Turkmenistan
179                                           Tuvalu
180                                          Uganda
181                                         Ukraine
182                               United Arab Emirates
183   United Kingdom of Great Britain and Northern I...
184                         United Republic of Tanzania
185                            United States of America
186                                         Uruguay
187                                      Uzbekistan
188                                         Vanuatu
189                 Venezuela (Bolivarian Republic of)
190                                         Viet Nam
191                                   Western Sahara
192                                           Yemen
193                                          Zambia
```

```
194                                                Zimbabwe
Name: Country, Length: 195, dtype: object
```

[33]:
```
df_dropping[['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.
```

[33]:

	Country	1980	1981	1982	\
0	Afghanistan	16	39	39	
1	Albania	1	0	0	
2	Algeria	80	67	71	
3	American Samoa	0	1	0	
4	Andorra	0	0	0	
5	Angola	1	3	6	
6	Antigua and Barbuda	0	0	0	
7	Argentina	368	426	626	
8	Armenia	0	0	0	
9	Australia	702	639	484	
10	Austria	234	238	201	
11	Azerbaijan	0	0	0	
12	Bahamas	26	23	38	
13	Bahrain	0	2	1	
14	Bangladesh	83	84	86	
15	Barbados	372	376	299	
16	Belarus	0	0	0	
17	Belgium	511	540	519	
18	Belize	16	27	13	
19	Benin	2	5	4	
20	Bhutan	0	0	0	
21	Bolivia (Plurinational State of)	44	52	42	
22	Bosnia and Herzegovina	0	0	0	
23	Botswana	10	1	3	
24	Brazil	211	220	192	
25	Brunei Darussalam	79	6	8	
26	Bulgaria	24	20	12	
27	Burkina Faso	2	1	3	
28	Burundi	0	0	0	
29	Cabo Verde	1	1	2	
..	...	...	...	...	
165	Suriname	15	10	21	
166	Swaziland	4	1	1	
167	Sweden	281	308	222	
168	Switzerland	806	811	634	
169	Syrian Arab Republic	315	419	409	
170	Tajikistan	0	0	0	
171	Thailand	56	53	113	

172	The former Yugoslav Republic of Macedonia	0	0	0
173	Togo	5	5	2
174	Tonga	2	4	7
175	Trinidad and Tobago	958	947	972
176	Tunisia	58	51	55
177	Turkey	481	874	706
178	Turkmenistan	0	0	0
179	Tuvalu	0	1	0
180	Uganda	13	16	17
181	Ukraine	0	0	0
182	United Arab Emirates	0	2	2
183	United Kingdom of Great Britain and Northern I...	22045	24796	20620
184	United Republic of Tanzania	635	832	621
185	United States of America	9378	10030	9074
186	Uruguay	128	132	146
187	Uzbekistan	0	0	0
188	Vanuatu	0	0	0
189	Venezuela (Bolivarian Republic of)	103	117	174
190	Viet Nam	1191	1829	2162
191	Western Sahara	0	0	0
192	Yemen	1	2	1
193	Zambia	11	17	11
194	Zimbabwe	72	114	102

	1983	1984	1985
0	47	71	340
1	0	0	0
2	69	63	44
3	0	0	0
4	0	0	0
5	6	4	3
6	0	42	52
7	241	237	196
8	0	0	0
9	317	317	319
10	117	127	165
11	0	0	0
12	12	21	28
13	1	1	3
14	81	98	92
15	244	265	285
16	0	0	0
17	297	183	181
18	21	37	26
19	3	4	3
20	0	1	0
21	49	38	44

```
22       0      0     0
23       3      7     4
24     139    145   130
25       2      2     4
26      33     11    24
27       2      3     2
28       0      1     2
29       0     11     1
..     ...    ...   ...
165     12      5    16
166      0     10     7
167    176    128   158
168    370    326   314
169    269    264   385
170      0      0     0
171     65     82    66
172      0      0     0
173      3      6     5
174      1      2     5
175    766    606   699
176     46     51    57
177    280    338   202
178      0      0     0
179      0      1     0
180     38     32    29
181      0      0     0
182      1      2     0
183  10015  10170  9564
184    474    473   460
185   7100   6661  6543
186    105     90    92
187      0      0     0
188      0      0     0
189    124    142   165
190   3404   7583  5907
191      0      0     0
192      6      0    18
193      7     16     9
194     44     32    29

[195 rows x 7 columns]
```

2 Select Row

There are main 3 ways to select rows:

`df.loc[label]`

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

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

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

[23]:
```
df_dropping.set_index('Country', inplace=True)
# tip: The opposite of set is reset. So to reset the index, we can use df_can.
 ↪reset_index()
```

[25]:
```
df_dropping.head(3)
```

[25]:

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203	2635	2004	58639
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223	856	702	560	716	561	539	620	603	15699
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626	4807	3623	4005	5393	4752	4325	3774	4331	69439

[3 rows x 38 columns]

3 Example: Let's view the number of immigrants from Japan (row 87) for the following scenarios:

1. The full row data (all columns)
2. For year 2013
3. For years 1980 to 1985

[26]:
```
# 1. the full row data (all columns)
print(df_dropping.loc['Japan'])

# alternate methods
#print(df_dropping.iloc[87])
#print(df_dropping[df_dropping.index == 'Japan'].T.squeeze())
```

```
Continent                 Asia
Region            Eastern Asia
DevName      Developed regions
1980                       701
1981                       756
1982                       598
1983                       309
1984                       246
1985                       198
1986                       248
1987                       422
1988                       324
1989                       494
1990                       379
1991                       506
1992                       605
1993                       907
1994                       956
1995                       826
1996                       994
1997                       924
1998                       897
1999                      1083
2000                      1010
2001                      1092
2002                       806
2003                       817
2004                       973
2005                      1067
2006                      1212
2007                      1250
2008                      1284
2009                      1194
2010                      1168
2011                      1265
2012                      1214
2013                       982
Total                    27707
Name: Japan, dtype: object
```

[27]:
```
# alternate methods
print(df_dropping.iloc[87])
```

```
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
```

[28]:
```
# 2. for year 2013
print(df_dropping.loc['Japan', 2013])

# alternate method
print(df_dropping.iloc[87, 36]) # year 2013 is the last column, with a␣
 ↪positional index of 36
```

```
982
982
```

[29]:
```
# 3. for years 1980 to 1985
print(df_dropping.loc['Japan', [1980, 1981, 1982, 1983, 1984, 1984]])

# alternate method
print(df_dropping.iloc[87, [3, 4, 5, 6, 7, 8]])
```

```
1980    701
1981    756
1982    598
1983    309
1984    246
1984    246
Name: Japan, dtype: object
1980    701
1981    756
1982    598
1983    309
1984    246
1985    198
Name: Japan, dtype: object
```

Column names that are integers (such as the years) might introduce some confusion. For example, when we are referencing the year 2013, one might confuse that when the 2013th positional index. To avoid this ambuigity, let's convert the column names into strings: '1980' to '2013'.

[30]:
```
#It will convert all column name into string
df_dropping.columns = list(map(str, df_dropping.columns))
```

[31]:
```
#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
```

[31]:
```
['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']
```

[32]:
```
df_dropping.head(9)
```

[32]:

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0
Angola	Africa	Middle Africa	Developing regions	1	3	6	6	4	3
Antigua and Barbuda	Latin America and the Caribbean	Caribbean	Developing regions	0	0	0	0	42	52
Argentina	Latin America and the Caribbean	South America	Developing regions	368	426	626	241	237	196
Armenia	Asia	Western Asia	Developing regions	0	0	0	0	0	0

Country	1986	...	2005	2006	2007	2008	2009	2010	2011

```
Country                 ...
Afghanistan             496  ...    3436  3009  2652  2111  1746  1758  2203
Albania                   1  ...    1223   856   702   560   716   561   539
Algeria                  69  ...    3626  4807  3623  4005  5393  4752  4325
American Samoa            0  ...       0     1     0     0     0     0     0
Andorra                   2  ...       0     1     1     0     0     0     0
Angola                    5  ...     295   184   106    76    62    61    39
Antigua and Barbuda      51  ...      24    32    15    32    38    27    37
Argentina               213  ...    1153   847   620   540   467   459   278
Armenia                   0  ...     224   218   198   205   267   252   236

                       2012  2013  Total
Country
Afghanistan            2635  2004  58639
Albania                 620   603  15699
Algeria                3774  4331  69439
American Samoa            0     0      6
Andorra                   1     1     15
Angola                   70    45   2113
Antigua and Barbuda      51    25    981
Argentina               263   282  19596
Armenia                 258   207   3310

[9 rows x 38 columns]
```

4 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).

[33]:
```
# 1. create the condition boolean series
condition = df_dropping['Continent'] == 'Asia'
print(condition)
```

```
Country
Afghanistan                                         True
Albania                                            False
Algeria                                            False
American Samoa                                     False
Andorra                                            False
Angola                                             False
Antigua and Barbuda                                False
Argentina                                          False
Armenia                                             True
Australia                                          False
Austria                                            False
Azerbaijan                                          True
```

Bahamas	False
Bahrain	True
Bangladesh	True
Barbados	False
Belarus	False
Belgium	False
Belize	False
Benin	False
Bhutan	True
Bolivia (Plurinational State of)	False
Bosnia and Herzegovina	False
Botswana	False
Brazil	False
Brunei Darussalam	True
Bulgaria	False
Burkina Faso	False
Burundi	False
Cabo Verde	False
	...
Suriname	False
Swaziland	False
Sweden	False
Switzerland	False
Syrian Arab Republic	True
Tajikistan	True
Thailand	True
The former Yugoslav Republic of Macedonia	False
Togo	False
Tonga	False
Trinidad and Tobago	False
Tunisia	False
Turkey	True
Turkmenistan	True
Tuvalu	False
Uganda	False
Ukraine	False
United Arab Emirates	True
United Kingdom of Great Britain and Northern Ireland	False
United Republic of Tanzania	False
United States of America	False
Uruguay	False
Uzbekistan	True
Vanuatu	False
Venezuela (Bolivarian Republic of)	False
Viet Nam	True
Western Sahara	False
Yemen	True
Zambia	False

```
Zimbabwe                                                False
Name: Continent, Length: 195, dtype: bool
```

5 BEST METHOD TO FILTER

[34]:
```
# we can pass mutliple criteria in the same line.
# let's filter for AreaNAme = Asia and RegName = Southern Asia

df_dropping[(df_dropping['Continent']=='Asia') &␣
 ↪(df_dropping['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
```

[34]:

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

```
Nepal                          540    511    581    561   1392   1129   1185
Pakistan                     13127  10124   8994   7217   6811   7468  11227
Sri Lanka                     4714   4123   4756   4547   4422   3309   3338

                              2013   Total
Country
Afghanistan                   2004   58639
Bangladesh                    3789   65568
Bhutan                         487    5876
India                        33087  691904
Iran (Islamic Republic of)   11291  175923
Maldives                         1      30
Nepal                         1308   10222
Pakistan                     12603  241600
Sri Lanka                     2394  148358

[9 rows x 38 columns]
```

[35]:
```
print('data dimensions:', df_dropping.shape)
print(df_dropping.columns)
df_dropping.head(2)
```

```
data dimensions: (195, 38)
Index(['Continent', 'Region', 'DevName', '1980', '1981', '1982', '1983',
       '1984', '1985', '1986', '1987', '1988', '1989', '1990', '1991', '1992',
       '1993', '1994', '1995', '1996', '1997', '1998', '1999', '2000', '2001',
       '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '2010',
       '2011', '2012', '2013', 'Total'],
      dtype='object')
```

[35]:
```
            Continent           Region             DevName  1980  1981  1982  \
Country
Afghanistan      Asia    Southern Asia  Developing regions    16    39    39
Albania        Europe  Southern Europe   Developed regions     1     0     0

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

             2010  2011  2012  2013  Total
Country
Afghanistan  1758  2203  2635  2004  58639
Albania       561   539   620   603  15699

[2 rows x 38 columns]
```

6 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().

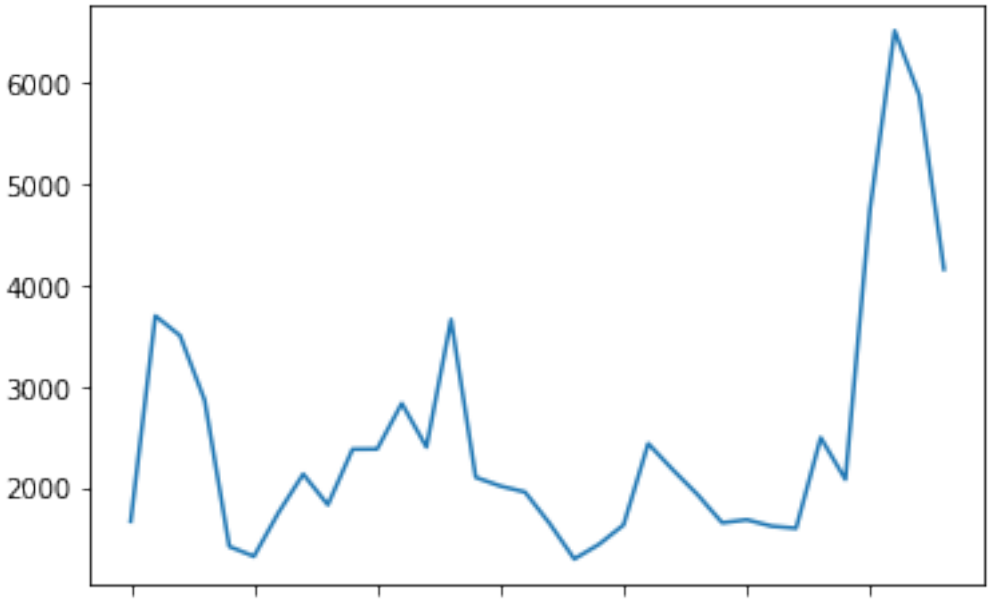
First, we will extract the data series for Haiti.

[49]:
```
haiti = df_dropping.loc['Haiti', years] # passing in years 1980 - 2013 to␣
↪exclude the 'total' column
haiti.head(10)
```

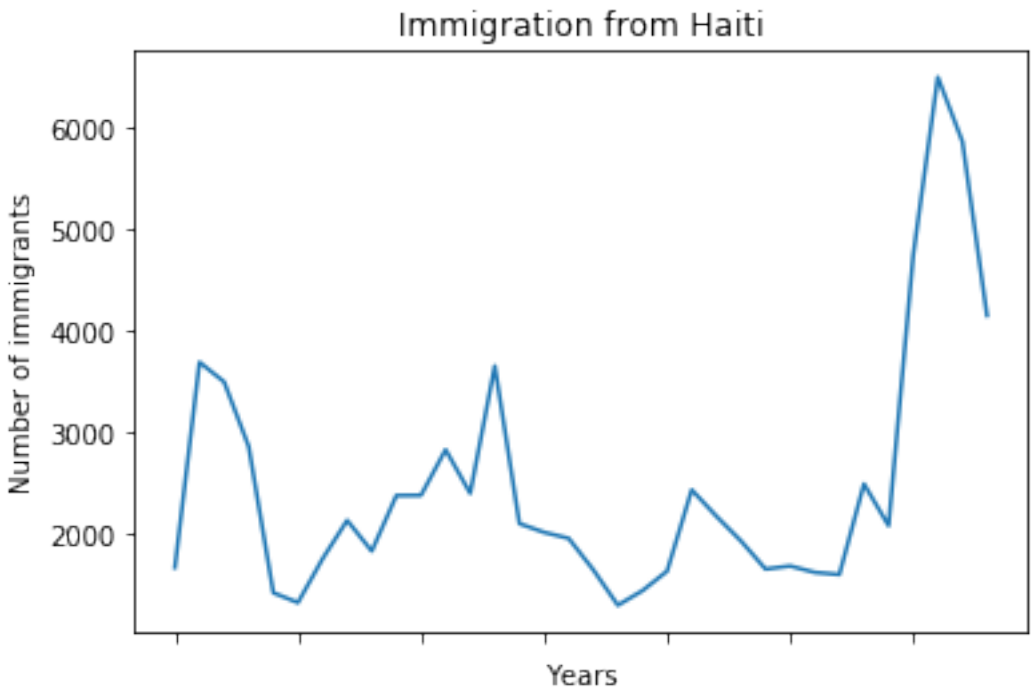
[49]:
```
1980    1666
1981    3692
1982    3498
1983    2860
1984    1418
1985    1321
1986    1753
1987    2132
1988    1829
1989    2377
Name: Haiti, dtype: object
```

[38]:
```
haiti.plot()
```

[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7d1cd88a90>
```

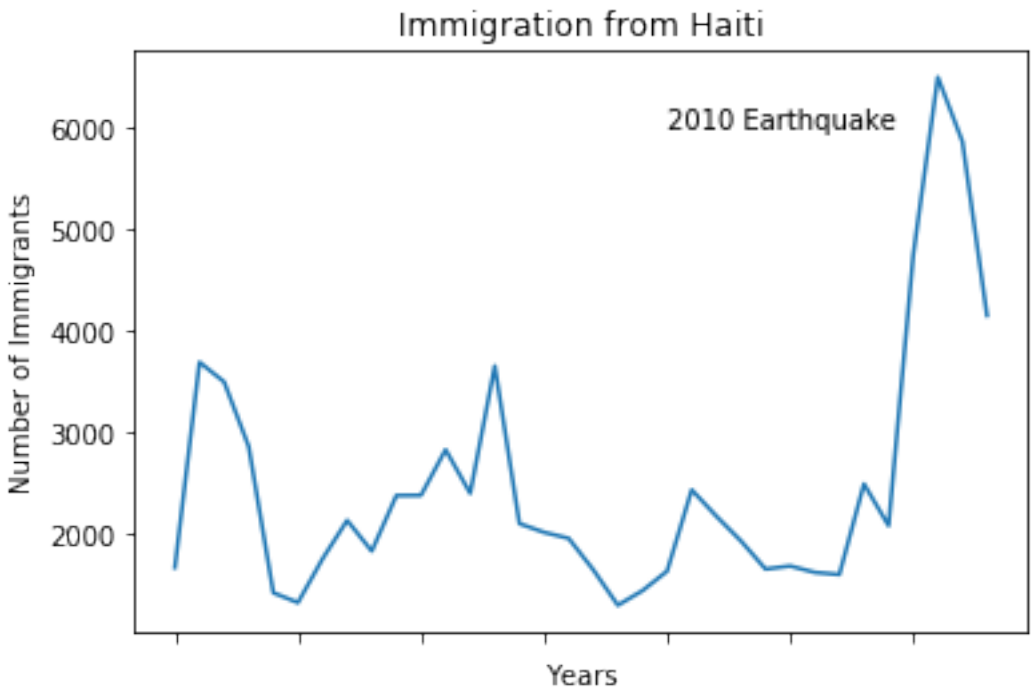


[39]:
```
#haiti.index = haiti.index.map(int) # let's change the index values of Haiti to␣
 ↪type integer for plotting
haiti.plot(kind='line')

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

plt.show()
```



7 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.

[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(20, 6000, '2010 Earthquake') # see note below, Here the we hv converted␣
 ↪colimns in string so we have to pass index numbed as 20 bcoz "2010 index␣
 ↪number is 20"


plt.show()
```



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

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

[46]:
```
df_CI = df_dropping.loc[['India', 'China'], years]
df_CI.head()
```

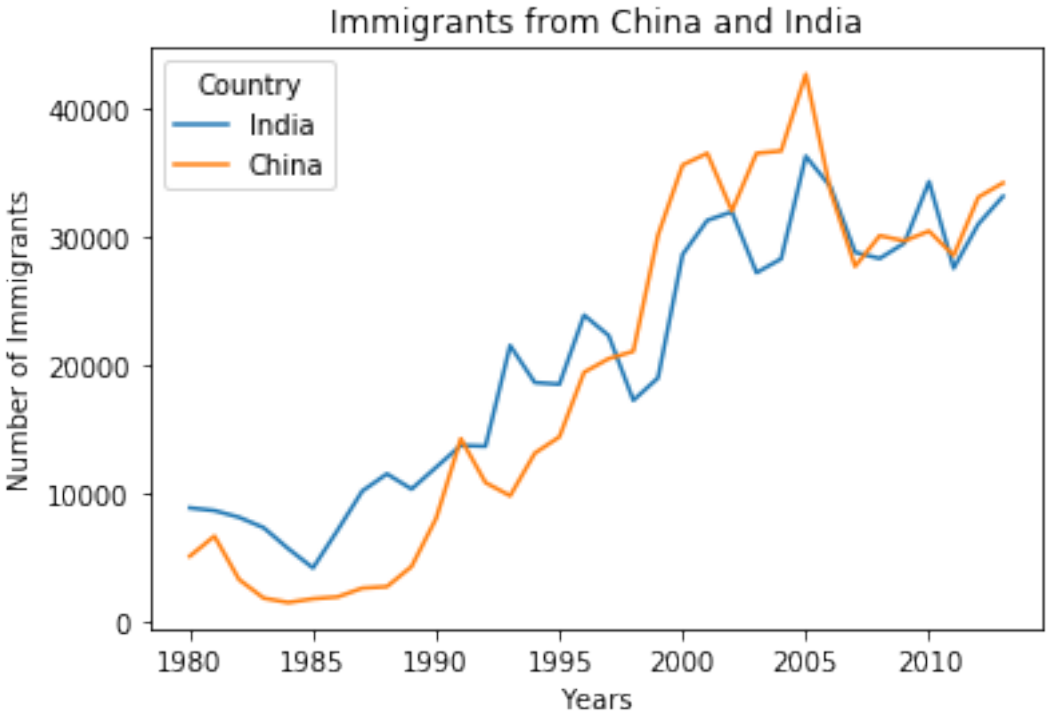
[46]:
```
         1980  1981  1982  1983  1984  1985  1986   1987   1988   1989  ...    \
Country                                                                 ...
India    8880  8670  8147  7338  5704  4211  7150  10189  11522  10343  ...
China    5123  6682  3308  1863  1527  1816  1960   2643   2758   4323  ...

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

[2 rows x 34 columns]
```

9 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.

[51]:
```
df_CI = df_CI.transpose()
df_CI.head()
```

[51]:
```
Country  India  China
1980      8880   5123
1981      8670   6682
1982      8147   3308
1983      7338   1863
1984      5704   1527
```

[54]:
```
df_CI.index = df_CI.index.map(int) # let's change the index values of df_CI to␣
 ↪type integer for plotting
df_CI.plot(kind='line')
plt.title('Immigrants from China and India')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



10 VERY IMPORTANT

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

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

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

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

[79]:
```
# Recall that we created a Total column that calculates the cumulative␣
 ↪immigration by country.
#\\ We will sort on this column to get our top 5 countries using pandas␣
 ↪sort_values() method.
df_top5 = df_dropping.sort_values(by='Total', ascending=False, axis=0,␣
 ↪inplace=True)
```

[80]:
```
df_top5 = df_dropping.head(5)
```

[81]:
```
df_top5 = df_top5[years].transpose()
```

[82]:
```
print(df_top5)
```

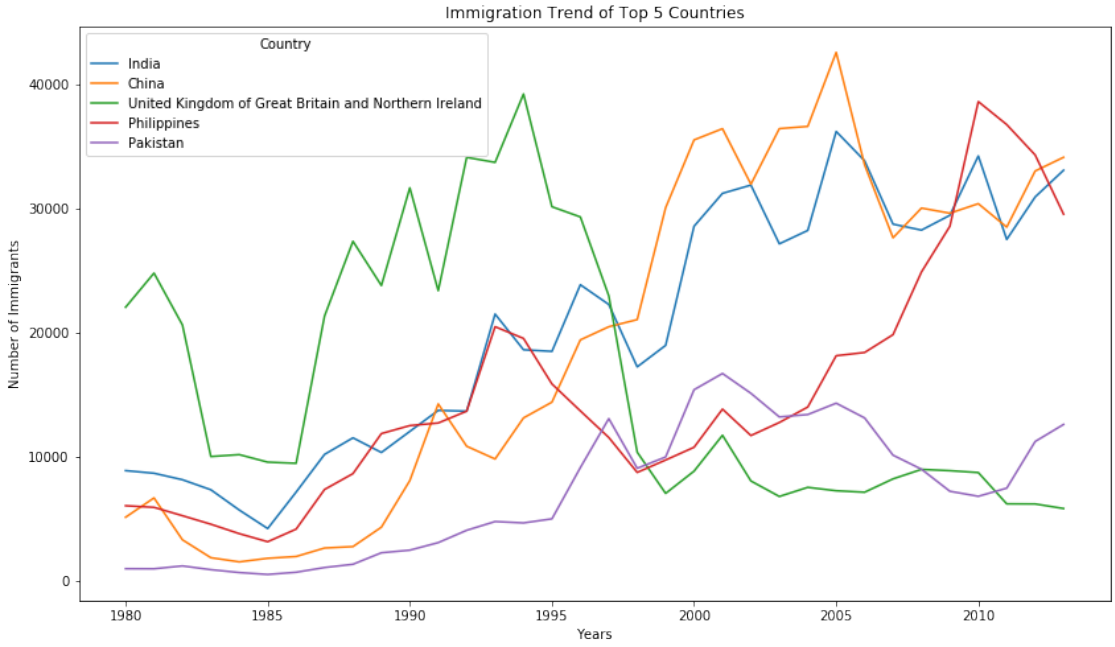
```
Country  India  China  United Kingdom of Great Britain and Northern Ireland  \
1980      8880   5123                                                22045
1981      8670   6682                                                24796
1982      8147   3308                                                20620
1983      7338   1863                                                10015
1984      5704   1527                                                10170
1985      4211   1816                                                 9564
1986      7150   1960                                                 9470
1987     10189   2643                                                21337
1988     11522   2758                                                27359
1989     10343   4323                                                23795
1990     12041   8076                                                31668
1991     13734  14255                                                23380
1992     13673  10846                                                34123
1993     21496   9817                                                33720
1994     18620  13128                                                39231
1995     18489  14398                                                30145
1996     23859  19415                                                29322
1997     22268  20475                                                22965
1998     17241  21049                                                10367
1999     18974  30069                                                 7045
```

2000	28572	35529	8840
2001	31223	36434	11728
2002	31889	31961	8046
2003	27155	36439	6797
2004	28235	36619	7533
2005	36210	42584	7258
2006	33848	33518	7140
2007	28742	27642	8216
2008	28261	30037	8979
2009	29456	29622	8876
2010	34235	30391	8724
2011	27509	28502	6204
2012	30933	33024	6195
2013	33087	34129	5827

Country	Philippines	Pakistan
1980	6051	978
1981	5921	972
1982	5249	1201
1983	4562	900
1984	3801	668
1985	3150	514
1986	4166	691
1987	7360	1072
1988	8639	1334
1989	11865	2261
1990	12509	2470
1991	12718	3079
1992	13670	4071
1993	20479	4777
1994	19532	4666
1995	15864	4994
1996	13692	9125
1997	11549	13073
1998	8735	9068
1999	9734	9979
2000	10763	15400
2001	13836	16708
2002	11707	15110
2003	12758	13205
2004	14004	13399
2005	18139	14314
2006	18400	13127
2007	19837	10124
2008	24887	8994
2009	28573	7217
2010	38617	6811
2011	36765	7468

```
2012            34315      11227
2013            29544      12603
```

[85]:
```
df_top5.index = df_top5.index.map(int) # let's change the index values of␣
 ↪df_top5 to type integer for plotting
df_top5.plot(kind='line', figsize=(14, 8))
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



[88]:
```
df_dropping.head()
```

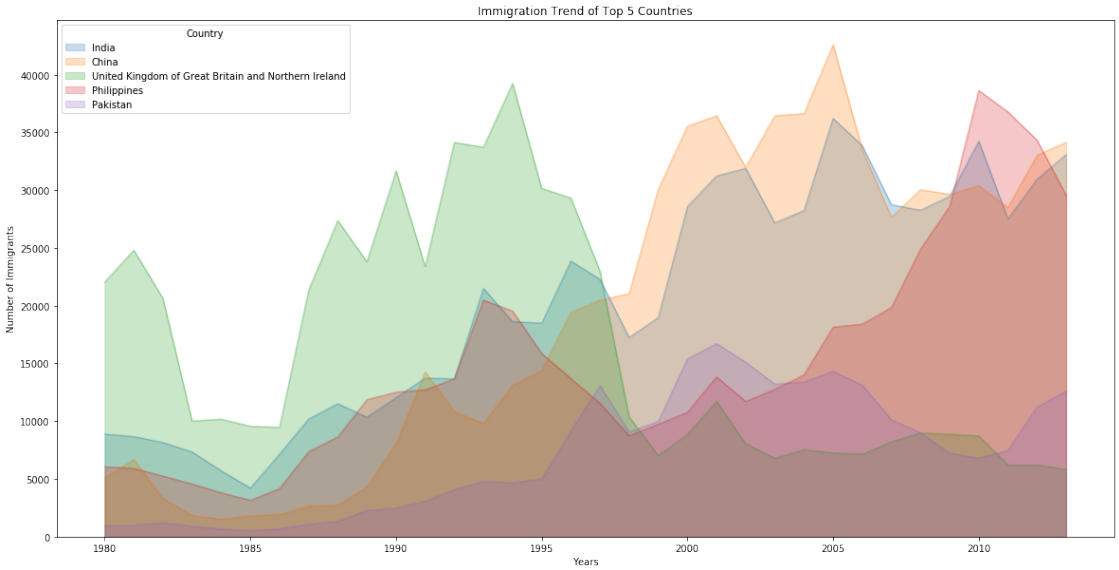
[88]:
```
                                                  Continent  \
Country
India                                                  Asia
China                                                  Asia
United Kingdom of Great Britain and Northern Ir...    Europe
Philippines                                            Asia
Pakistan                                               Asia

                                                        Region  \
Country
India                                             Southern Asia
China                                              Eastern Asia
United Kingdom of Great Britain and Northern Ir...  Northern Europe
Philippines                                       South-Eastern Asia
```

```
Pakistan                                                Southern Asia

                                                              DevName   1980  \
Country
India                                                 Developing regions   8880
China                                                 Developing regions   5123
United Kingdom of Great Britain and Northern Ir...     Developed regions  22045
Philippines                                           Developing regions   6051
Pakistan                                              Developing regions    978

                                                       1981   1982   1983  \
Country
India                                                  8670   8147   7338
China                                                  6682   3308   1863
United Kingdom of Great Britain and Northern Ir...    24796  20620  10015
Philippines                                            5921   5249   4562
Pakistan                                                972   1201    900

                                                       1984  1985  1986   ...    \
Country                                                                   ...
India                                                  5704  4211  7150   ...
China                                                  1527  1816  1960   ...
United Kingdom of Great Britain and Northern Ir...    10170  9564  9470   ...
Philippines                                            3801  3150  4166   ...
Pakistan                                                668   514   691   ...

                                                       2005   2006   2007  \
Country
India                                                 36210  33848  28742
China                                                 42584  33518  27642
United Kingdom of Great Britain and Northern Ir...     7258   7140   8216
Philippines                                           18139  18400  19837
Pakistan                                              14314  13127  10124

                                                       2008   2009   2010  \
Country
India                                                 28261  29456  34235
China                                                 30037  29622  30391
United Kingdom of Great Britain and Northern Ir...     8979   8876   8724
Philippines                                           24887  28573  38617
Pakistan                                               8994   7217   6811

                                                       2011   2012   2013  \
Country
India                                                 27509  30933  33087
China                                                 28502  33024  34129
United Kingdom of Great Britain and Northern Ir...     6204   6195   5827
```

```
Philippines                                             36765  34315  29544
Pakistan                                                 7468  11227  12603

                                                         Total
Country
India                                                   691904
China                                                   659962
United Kingdom of Great Britain and Northern Ir...      551500
Philippines                                             511391
Pakistan                                                241600

[5 rows x 38 columns]
```

12 Area Plots

[92]:
```
#The unstacked plot has a default transparency (alpha value) at 0.5. We can␣
 ↪modify this value by passing in the alpha parameter
df_top5.index = df_top5.index.map(int) # let's change the index values of␣
 ↪df_top5 to type integer for plotting
df_top5.plot(kind='area',
             stacked=False,
             alpha = 0.25,
             figsize=(20, 10), # pass a tuple (x, y) size
             )

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

plt.show()
```



13 Question: Use the scripting layer to create a stacked area plot of the 5 countries that contributed the least to immigration to Canada from 1980 to 2013. Use a transparency value of 0.45.

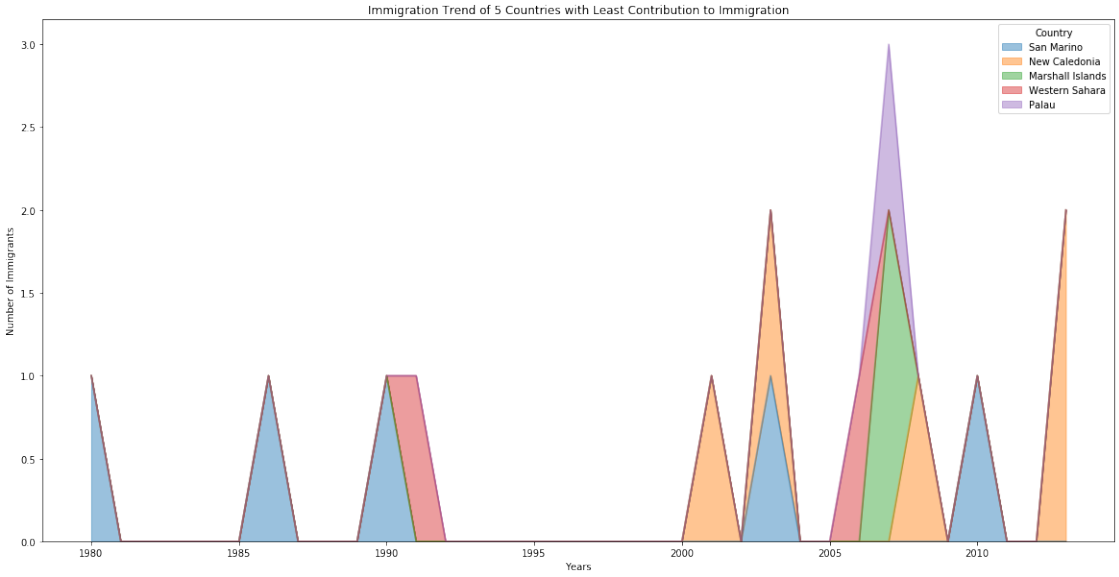
[93]:
```
df_least5 = df_dropping.tail(5)
```

[94]:
```
df_least5 = df_least5[years].transpose()
df_least5.head()
```

[94]:

Country	San Marino	New Caledonia	Marshall Islands	Western Sahara	Palau
1980	1	0	0	0	0
1981	0	0	0	0	0
1982	0	0	0	0	0
1983	0	0	0	0	0
1984	0	0	0	0	0

[97]:
```
df_least5.index = df_least5.index.map(int) # let's change the index values of␣
↪df_least5 to type integer for plotting
df_least5.plot(kind='area', alpha=0.45, figsize=(20, 10))
plt.title('Immigration Trend of 5 Countries with Least Contribution to␣
↪Immigration')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



14 PIE PLOT

[104]:
```
df_continents = df_dropping.groupby('Continent', axis=0).sum()

# note: the output of the groupby method is a `groupby' object.
# we can not use it further until we apply a function (eg .sum())
print(type(df_dropping.groupby('Continent', axis=0)))

df_continents.head()
```

```
<class 'pandas.core.groupby.groupby.DataFrameGroupBy'>
```

[104]:

Continent	1980	1981	1982	1983	1984	1985	\
Africa	3951	4363	3819	2671	2639	2650	
Asia	31025	34314	30214	24696	27274	23850	
Europe	39760	44802	42720	24638	22287	20844	
Latin America and the Caribbean	13081	15215	16769	15427	13678	15171	
Northern America	9378	10030	9074	7100	6661	6543	

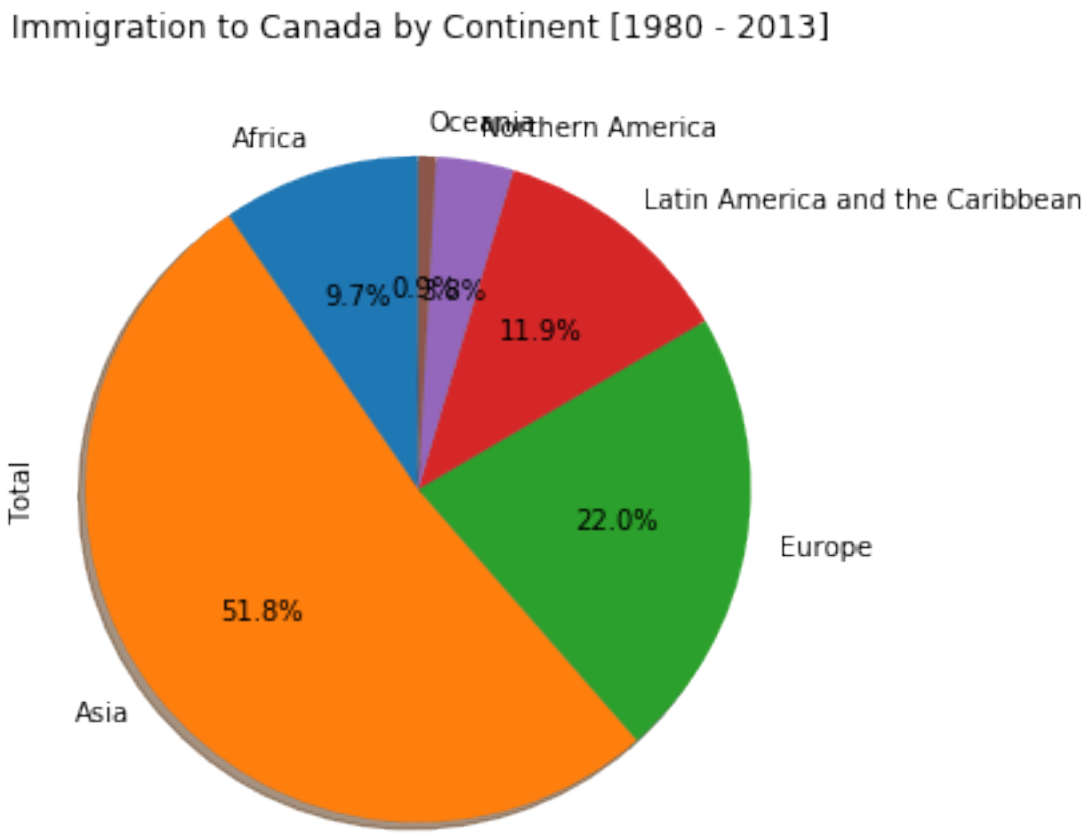
Continent	1986	1987	1988	1989	...	2005	\
					...		
Africa	3782	7494	7552	9894	...	27523	
Asia	28739	43203	47454	60256	...	159253	
Europe	24370	46698	54726	60893	...	35955	
Latin America and the Caribbean	21179	28471	21924	25060	...	24747	
Northern America	7074	7705	6469	6790	...	8394	

```
                                  2006    2007    2008    2009    2010  \
Continent
Africa                           29188   28284   29890   34534   40892
Asia                            149054  133459  139894  141434  163845
Europe                           33053   33495   34692   35078   33425
Latin America and the Caribbean  24676   26011   26547   26867   28818
Northern America                  9613    9463   10190    8995    8142

                                  2011    2012    2013    Total
Continent
Africa                           35441   38083   38543   618948
Asia                            146894  152218  155075  3317794
Europe                           26778   29177   28691  1410947
Latin America and the Caribbean  27856   27173   24950   765148
Northern America                  7677    7892    8503   241142

[5 rows x 35 columns]
```

15 Step 2: Plot the data. We will pass in kind = 'pie' keyword, along with the following additional parameters:

autopct - is a string or function used to label the wedges with their numeric value. The label will be placed inside the wedge. If it is a format string, the label will be fmt%pct. startangle - rotates the start of the pie chart by angle degrees counterclockwise from the x-axis. shadow - Draws a shadow beneath the pie (to give a 3D feel).

[101]:
```
# autopct create %, start angle represent starting point
df_continents['Total'].plot(kind='pie',
                            figsize=(5, 6),
                            autopct='%1.1f%%', # add in percentages
                            startangle=90,     # start angle 90ř (Africa)
                            shadow=True,# add shadow
                            )

plt.title('Immigration to Canada by Continent [1980 - 2013]')
plt.axis('equal') # Sets the pie chart to look like a circle.

plt.show()
```

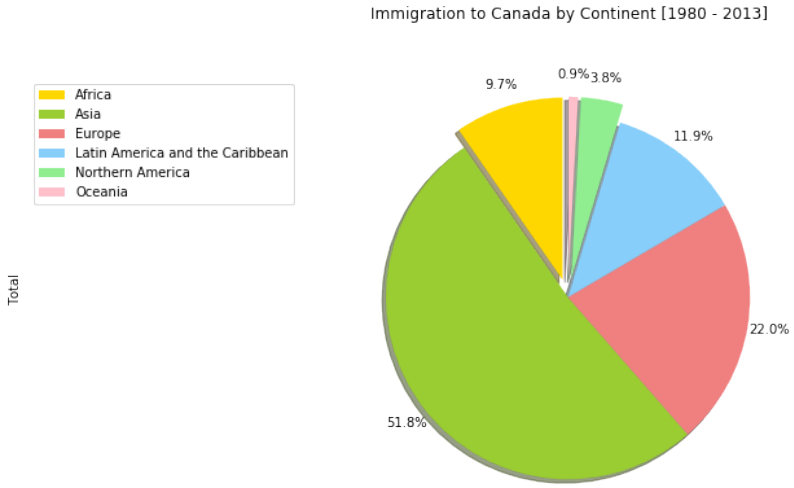


16 The above visual is not very clear, the numbers and text overlap in some instances. Let's make a few modifications to improve the visuals:

Remove the text labels on the pie chart by passing in legend and add it as a seperate legend using plt.legend(). Push out the percentages to sit just outside the pie chart by passing in pctdistance parameter. Pass in a custom set of colors for continents by passing in colors parameter. Explode the pie chart to emphasize the lowest three continents (Africa, North America, and Latin America and Carribbean) by pasing in explode parameter.

[106]:
```
colors_list = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue',␣
↪'lightgreen', 'pink']
explode_list = [0.1, 0, 0, 0, 0.1, 0.1] # ratio for each continent with which to␣
↪offset each wedge.
#We are choosing ratio from row wise like 1st Africa, 2nd Asia, 3rd Europe....etc
```

```
df_continents['Total'].plot(kind='pie',
                            figsize=(15, 6),
                            autopct='%1.1f%%',
                            startangle=90,
                            shadow=True,
                            labels=None,         # turn off labels on pie chart
                            pctdistance=1.12,    # the ratio between the center␣
 ↪of each pie slice and the start of the text generated by autopct
                            colors=colors_list,  # add custom colors
                            explode=explode_list # 'explode' lowest 3 continents
                            )

# scale the title up by 12% to match pctdistance
plt.title('Immigration to Canada by Continent [1980 - 2013]', y=1.12)

plt.axis('equal')

# add legend
plt.legend(labels=df_continents.index, loc='upper left')

plt.show()
```



17 Box Plots

18 A box plot is a way of statistically representing the distribution of the data through five main dimensions:

Minimun: Smallest number in the dataset. First quartile: Middle number between the minimum and the median. Second quartile (Median): Middle number of the (sorted) dataset. Third quartile:

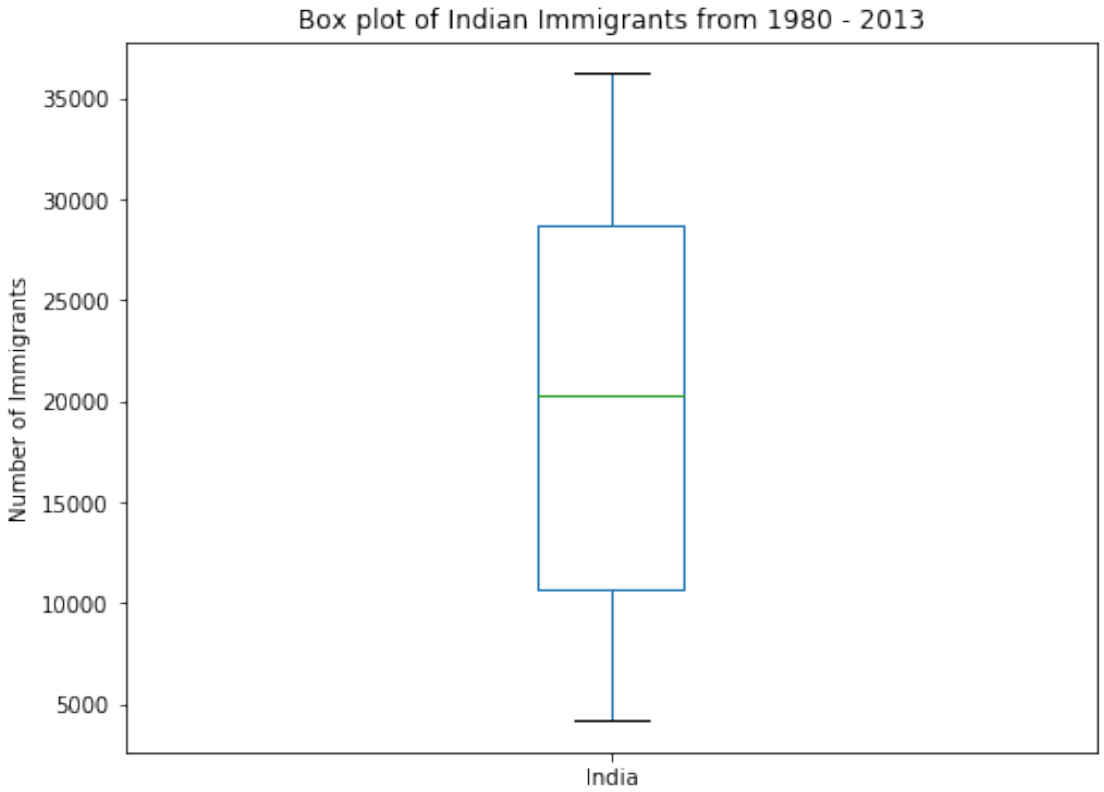
Middle number between median and maximum. Maximum: Highest number in the dataset.

19 Question: Let's plot the box plot for the INDIAN immigrants between 1980 - 2013.

[107]:
```
# to get a dataframe, place extra square brackets around 'Japan'.
df_india = df_dropping.loc[['India'], years].transpose()
df_india.head()
```

[107]:
```
Country  India
1980      8880
1981      8670
1982      8147
1983      7338
1984      5704
```

[109]:
```
df_india.plot(kind='box', figsize=(8, 6))

plt.title('Box plot of Indian Immigrants from 1980 - 2013')
plt.ylabel('Number of Immigrants')

plt.show()
```



20 We can immediately make a few key observations from the plot above:

The minimum number of immigrants is around 4000 (min), maximum number is around 36000(max), and median number of immigrants is around 20000 (median).

25% of the years for period 1980 - 2013 had an annual immigrant count of ~900 or little more (First quartile).

75% of the years for period 1980 - 2013 had an annual immigrant count of ~27000 or little more (Third quartile).

[111]:
```
df_india.describe()
```

[111]:
```
Country         India
count       34.000000
mean     20350.117647
std      10007.342579
min       4211.000000
25%      10637.750000
50%      20235.000000
75%      28699.500000
max      36210.000000
```

21 Subplots

Often times we might want to plot multiple plots within the same figure. For example, we might want to perform a side by side comparison of the box plot with the line plot of China and India's immigration.

To visualize multiple plots together, we can create a figure (overall canvas) and divide it into subplots, each containing a plot. With subplots, we usually work with the artist layer instead of the scripting layer.

Typical syntax is :

```
fig = plt.figure() # create figure
ax = fig.add_subplot(nrows, ncols, plot_number) # create subplots
```

Where

nrows and ncols are used to notionally split the figure into (nrows * ncols) sub-axes, plot_number is used to identify the particular subplot that this function is to create within the notional grid. plot_number starts at 1, increments across rows first and has a maximum of nrows * ncols as shown below.

22 *For sub plot see the clear picture present in my won data science sheet.* **# If possible extract more information regarding development of country, etc**

23 OVER-

[ ]: