pandas Basics

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

In [3]:
```
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
```

Let's download and import our primary Canadian Immigration dataset using *pandas* `read_excel()` method. Normally, before we can do that, we would need to download a module which *pandas* requires to read in excel files. This module is **xlrd**. For your convenience, we have pre-installed this module, so you would not have to worry about that. Otherwise, you would need to run the following line of code to install the **xlrd** module:

```
!conda install -c anaconda xlrd --yes
```

Now we are ready to read in our data.

In [4]:
```
df_can = pd.read_excel('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data_Files/Canada.xlsx',
                       sheet_name='Canada by Citizenship',
                       skiprows=range(20),
                       skipfooter=2)

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

```
Data read into a pandas dataframe!
```

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

In [5]:
```
df_can.head()
# tip: You can specify the number of rows you'd like to see as follows:
 df_can.head(10)
```

Out[5]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0

5 rows × 43 columns

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

In [6]:
```
df_can.tail()
```

Out[6]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980
190	Immigrants	Foreigners	Viet Nam	935	Asia	920	South-Eastern Asia	902	Developing regions	1191
191	Immigrants	Foreigners	Western Sahara	903	Africa	912	Northern Africa	902	Developing regions	0
192	Immigrants	Foreigners	Yemen	935	Asia	922	Western Asia	902	Developing regions	1
193	Immigrants	Foreigners	Zambia	903	Africa	910	Eastern Africa	902	Developing regions	11

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980
194	Immigrants	Foreigners	Zimbabwe	903	Africa	910	Eastern Africa	902	Developing regions	72

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.

In [7]:
```
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
```

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

In [8]:
```
df_can.columns.values
```

Out[8]:
```
array(['Type', 'Coverage', 'OdName', 'AREA', 'AreaName', 'REG', 'RegNam
e',
       'DEV', 'DevName', 1980, 1981, 1982, 1983, 1984, 1985, 1986, 198
7,
       1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 199
8,
       1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 200
9,
       2010, 2011, 2012, 2013], dtype=object)
```

Similarly, to get the list of indicies we use the `.index` parameter.

In [9]:
```
df_can.index.values
```

Out[9]:
```
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, 19
4])
```

Note: The default type of index and columns is NOT list.

In [10]:
```
print(type(df_can.columns))
print(type(df_can.index))
```

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

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

In [11]:
```
df_can.columns.tolist()
df_can.index.tolist()

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

```
<class 'list'>
<class 'list'>
```

To view the dimensions of the dataframe, we use the `.shape` parameter.

In [12]:
```
# size of dataframe (rows, columns)
df_can.shape
```

Out[12]:
```
(195, 43)
```

Note: The main types stored in *pandas* objects are *float*, *int*, *bool*, *datetime64[ns]* and *datetime64[ns, tz] (in >= 0.17.0)*, *timedelta[ns]*, *category (in >= 0.15.0)*, and *object* (string). In addition these dtypes have item sizes, e.g. int64 and int32.

Let's clean the data set to remove a few unnecessary columns. We can use *pandas* `drop()` method as follows:

In [13]:
```
# in pandas axis=0 represents rows (default) and axis=1 represents colu
mns.
df_can.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=Tru
e)
df_can.head(2)
```

Out[13]:

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	...	2004
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	2978
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	1450

2 rows × 38 columns

Let's rename the columns so that they make sense. We can use `rename()` method by passing in a dictionary of old and new names as follows:

In [14]:
```
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'Reg
Name':'Region'}, inplace=True)
df_can.columns
```

Out[14]:
```
Index([  'Country', 'Continent',    'Region',   'DevName',        1980,
              1981,        1982,        1983,        1984,        1985,
              1986,        1987,        1988,        1989,        1990,
              1991,        1992,        1993,        1994,        1995,
              1996,        1997,        1998,        1999,        2000,
              2001,        2002,        2003,        2004,        2005,
              2006,        2007,        2008,        2009,        2010,
              2011,        2012,        2013],
      dtype='object')
```

We will also add a 'Total' column that sums up the total immigrants by country over the entire period 1980 - 2013, as follows:

In [15]:
```
df_can['Total'] = df_can.sum(axis=1)
```

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

In [16]:
```
df_can.isnull().sum()
```

Out[16]:
```
Country      0
Continent    0
Region       0
DevName      0
1980         0
1981         0
1982         0
1983         0
1984         0
1985         0
1986         0
1987         0
1988         0
1989         0
```

```
1990        0
1991        0
1992        0
1993        0
1994        0
1995        0
1996        0
1997        0
1998        0
1999        0
2000        0
2001        0
2002        0
2003        0
2004        0
2005        0
2006        0
2007        0
2008        0
2009        0
2010        0
2011        0
2012        0
2013        0
Total       0
dtype: int64
```

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

In [17]:
```
df_can.describe()
```

Out[17]:

	1980	1981	1982	1983	1984	1985	
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.
mean	508.394872	566.989744	534.723077	387.435897	376.497436	358.861538	441.

	1980	1981	1982	1983	1984	1985	
std	1949.588546	2152.643752	1866.997511	1204.333597	1198.246371	1079.309600	1225.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
50%	13.000000	10.000000	11.000000	12.000000	13.000000	17.000000	18.
75%	251.500000	295.500000	275.000000	173.000000	181.000000	197.000000	254.
max	22045.000000	24796.000000	20620.000000	10015.000000	10170.000000	9564.000000	9470.

8 rows × 35 columns

pandas Intermediate: Indexing and Selection (slicing)

Select Column

There are two ways to filter on a column name:

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

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

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

```
df['column']
        (returns series)
```

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

Example: Let's try filtering on the list of countries ('Country').

In [18]:
```
df_can.Country  # returns a series
```

Out[18]:
```
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
```

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

In [19]:
```
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 stri
ng later on.
```

Out[19]:

	Country	1980	1981	1982	1983	1984	1985
0	Afghanistan	16	39	39	47	71	340
1	Albania	1	0	0	0	0	0
2	Algeria	80	67	71	69	63	44
3	American Samoa	0	1	0	0	0	0
4	Andorra	0	0	0	0	0	0
5	Angola	1	3	6	6	4	3
6	Antigua and Barbuda	0	0	0	0	42	52
7	Argentina	368	426	626	241	237	196
8	Armenia	0	0	0	0	0	0
9	Australia	702	639	484	317	317	319
10	Austria	234	238	201	117	127	165
11	Azerbaijan	0	0	0	0	0	0
12	Bahamas	26	23	38	12	21	28
13	Bahrain	0	2	1	1	1	3
14	Bangladesh	83	84	86	81	98	92
15	Barbados	372	376	299	244	265	285
16	Belarus	0	0	0	0	0	0
17	Belgium	511	540	519	297	183	181
18	Belize	16	27	13	21	37	26
19	Benin	2	5	4	3	4	3
20	Bhutan	0	0	0	0	1	0
21	Bolivia (Plurinational State of)	44	52	42	49	38	44

22	Bosnia and Herzegovina	0	0	0	0	0	0

	Country	1980	1981	1982	1983	1984	1985
23	Botswana	10	1	3	3	7	4
24	Brazil	211	220	192	139	145	130
25	Brunei Darussalam	79	6	8	2	2	4
26	Bulgaria	24	20	12	33	11	24
27	Burkina Faso	2	1	3	2	3	2
28	Burundi	0	0	0	0	1	2
29	Cabo Verde	1	1	2	0	11	1
...	...	...	...	...	...	...	...
165	Suriname	15	10	21	12	5	16
166	Swaziland	4	1	1	0	10	7
167	Sweden	281	308	222	176	128	158
168	Switzerland	806	811	634	370	326	314
169	Syrian Arab Republic	315	419	409	269	264	385
170	Tajikistan	0	0	0	0	0	0
171	Thailand	56	53	113	65	82	66
172	The former Yugoslav Republic of Macedonia	0	0	0	0	0	0
173	Togo	5	5	2	3	6	5
174	Tonga	2	4	7	1	2	5
175	Trinidad and Tobago	958	947	972	766	606	699
176	Tunisia	58	51	55	46	51	57
177	Turkey	481	874	706	280	338	202
178	Turkmenistan	0	0	0	0	0	0
179	Tuvalu	0	1	0	0	1	0

180	Uganda	13	16	17	38	32	29
181	Ukraine	0	0	0	0	0	0
	Country	**1980**	**1981**	**1982**	**1983**	**1984**	**1985**
182	United Arab Emirates	0	2	2	1	2	0
183	United Kingdom of Great Britain and Northern I...	22045	24796	20620	10015	10170	9564
184	United Republic of Tanzania	635	832	621	474	473	460
185	United States of America	9378	10030	9074	7100	6661	6543
186	Uruguay	128	132	146	105	90	92
187	Uzbekistan	0	0	0	0	0	0
188	Vanuatu	0	0	0	0	0	0
189	Venezuela (Bolivarian Republic of)	103	117	174	124	142	165
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 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.

In [20]:
```
df_can.set_index('Country', inplace=True)
# tip: The opposite of set is reset. So to reset the index, we can use
 df_can.reset_index()
```

In [21]:
```
df_can.head(3)
```

Out[21]:

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626

3 rows × 38 columns

In [22]:
```
# optional: to remove the name of the index
df_can.index.name = None
```

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

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

In [23]:
```
# 1. the full row data (all columns)
print(df_can.loc['Japan'])

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

```
Continent                 Asia
Region             Eastern Asia
DevName      Developed regions
1980                       701
1981                       756
1982                       598
1983                       309
1984                       246
1985                       198
1986                       248
1987                       422
1988                       324
1989                       494
1990                       379
1991                       506
1992                       605
1993                       907
1994                       956
1995                       826
1996                       994
1997                       924
1998                       897
1999                      1083
2000                      1010
2001                      1092
2002                       806
2003                       817
2004                       973
2005                      1067
2006                      1212
2007                      1250
2008                      1284
```

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

```
2009                    1194
2010                    1168
2011                    1265
2012                    1214
2013                     982
Total                  27707
Name: Japan, dtype: object
```

In [24]:
```
# 2. for year 2013
print(df_can.loc['Japan', 2013])

# alternate method
print(df_can.iloc[87, 36]) # year 2013 is the last column, with a posit
ional index of 36
```

```
982
982
```

In [25]:
```
# 3. for years 1980 to 1985
print(df_can.loc['Japan', [1980, 1981, 1982, 1983, 1984, 1984]])
print(df_can.iloc[87, [3, 4, 5, 6, 7, 8]])
```

```
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'.

In [26]:
```
df_can.columns = list(map(str, df_can.columns))
# [print (type(x)) for x in df_can.columns.values] #<-- uncomment to ch
eck type of column headers
```

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

In [27]:
```
# useful for plotting later on
years = list(map(str, range(1980, 2014)))
years
```

Out[27]:
```
['1980',
 '1981',
 '1982',
 '1983',
 '1984',
 '1985',
 '1986',
 '1987',
 '1988',
 '1989',
 '1990',
 '1991',
 '1992',
 '1993',
 '1994',
 '1995',
 '1996',
 '1997',
 '1998',
 '1999',
 '2000',
```

```
 '2001',
 '2002',
 '2003',
 '2004',
 '2005',
 '2006',
 '2007',
 '2008',
 '2009',
 '2010',
 '2011',
 '2012',
 '2013']
```

Filtering based on a criteria

To filter the dataframe based on a condition, we simply pass the condition as a boolean vector.

For example, Let's filter the dataframe to show the data on Asian countries (AreaName = Asia).

In [28]:
```
# 1. create the condition boolean series
condition = df_can['Continent'] == 'Asia'
print(condition)
```

```
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
```

In [29]:
```
# 2. pass this condition into the dataFrame
df_can[condition]
```

Out[29]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	
Armenia	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	
Azerbaijan	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	
Bahrain	Asia	Western Asia	Developing regions	0	2	1	1	1	3	0	...	
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	...	
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	0	...	
Brunei Darussalam	Asia	South-Eastern Asia	Developing regions	79	6	8	2	2	4	12	...	
Cambodia	Asia	South-Eastern Asia	Developing regions	12	19	26	33	10	7	8	...	
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	...	

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2
China, Hong Kong Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...	
China, Macao Special Administrative Region	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...	
Cyprus	Asia	Western Asia	Developing regions	132	128	84	46	46	43	48	...	
Democratic People's Republic of Korea	Asia	Eastern Asia	Developing regions	1	1	3	1	4	3	0	...	
Georgia	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36
Indonesia	Asia	South-Eastern Asia	Developing regions	186	178	252	115	123	100	127	...	
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	...	5
Iraq	Asia	Western Asia	Developing regions	262	245	260	380	428	231	265	...	2
Israel	Asia	Western Asia	Developing regions	1403	1711	1334	541	446	680	1212	...	2
Japan	Asia	Eastern Asia	Developed regions	701	756	598	309	246	198	248	...	1
Jordan	Asia	Western Asia	Developing regions	177	160	155	113	102	179	181	...	1
Kazakhstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2
Kuwait	Asia	Western Asia	Developing regions	1	0	8	2	1	4	4	...	
Kyrgyzstan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	
Lao People's Democratic Republic	Asia	South-Eastern Asia	Developing regions	11	6	16	16	7	17	21	...	
Lebanon	Asia	Western Asia	Developing regions	1409	1119	1159	789	1253	1683	2576	...	3
Malaysia	Asia	South-Eastern Asia	Developing regions	786	816	813	448	384	374	425	...	
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	...	
Mongolia	Asia	Eastern Asia	Developing regions	0	0	0	0	0	0	0	...	
Myanmar	Asia	South-Eastern Asia	Developing regions	80	62	46	31	41	23	18	...	
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	13	...	
Oman	Asia	Western Asia	Developing regions	0	0	0	8	0	0	0	...	
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...	18
Qatar	Asia	Western Asia	Developing regions	0	0	0	0	0	0	1	...	
Republic of Korea	Asia	Eastern Asia	Developing regions	1011	1456	1572	1081	847	962	1208	...	5

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2
Saudi Arabia	Asia	Western Asia	Developing regions	0	0	1	4	1	2	5	...	
Singapore	Asia	South-Eastern Asia	Developing regions	241	301	337	169	128	139	205	...	
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	...	4
State of Palestine	Asia	Western Asia	Developing regions	0	0	0	0	0	0	0	...	
Syrian Arab Republic	Asia	Western Asia	Developing regions	315	419	409	269	264	385	493	...	1
Tajikistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	
Thailand	Asia	South-Eastern Asia	Developing regions	56	53	113	65	82	66	78	...	
Turkey	Asia	Western Asia	Developing regions	481	874	706	280	338	202	257	...	2
Turkmenistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	
United Arab Emirates	Asia	Western Asia	Developing regions	0	2	2	1	2	0	5	...	
Uzbekistan	Asia	Central Asia	Developing regions	0	0	0	0	0	0	0	...	
Viet Nam	Asia	South-Eastern Asia	Developing regions	1191	1829	2162	3404	7583	5907	2741	...	1
Yemen	Asia	Western Asia	Developing regions	1	2	1	6	0	18	7	...	

49 rows × 38 columns

In [30]:
```
# we can pass mutliple criteria in the same line.
```

```
# let's filter for AreaNAme = Asia and RegName = Southern Asia

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

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

Out[30]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	200
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	343
Bangladesh	Asia	Southern Asia	Developing regions	83	84	86	81	98	92	486	...	417
Bhutan	Asia	Southern Asia	Developing regions	0	0	0	0	1	0	0	...	
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	3621
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	...	583
Maldives	Asia	Southern Asia	Developing regions	0	0	0	1	0	0	0	...	
Nepal	Asia	Southern Asia	Developing regions	1	1	6	1	2	4	13	...	60
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	1431
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	...	493

9 rows × 38 columns

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

In [31]:
```
print('data dimensions:', df_can.shape)
print(df_can.columns)
df_can.head(2)
```

```
data dimensions: (195, 38)
Index(['Continent', 'Region', 'DevName', '1980', '1981', '1982', '198
3',
       '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')
```

Out[31]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	200
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	343
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	122

2 rows × 38 columns

Visualizing Data using Matplotlib

Matplotlib: Standard Python Visualization Library

The primary plotting library we will explore in the course is Matplotlib. As mentioned on their website:

> Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

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

Matplotlib.Pyplot

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

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

In [32]:
```
# we are using the inline backend
%matplotlib inline

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

*optional: check if Matplotlib is loaded.

In [33]:
```
print ('Matplotlib version: ', mpl.__version__) # >= 2.0.0
```

```
Matplotlib version:  3.0.3
```

*optional: apply a style to Matplotlib.

In [34]:
```
print(plt.style.available)
mpl.style.use(['ggplot']) # optional: for ggplot-like style
```

```
['seaborn-dark', 'seaborn-darkgrid', 'seaborn-ticks', 'fivethirtyeigh
t', 'seaborn-whitegrid', 'classic', '_classic_test', 'fast', 'seaborn-t
alk', 'seaborn-dark-palette', 'seaborn-bright', 'seaborn-pastel', 'gray
scale', 'seaborn-notebook', 'ggplot', 'seaborn-colorblind', 'seaborn-mu
ted', 'seaborn', 'Solarize_Light2', 'seaborn-paper', 'bmh', 'tableau-co
lorblind10', 'seaborn-white', 'dark_background', 'seaborn-poster', 'sea
born-deep']
```

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.

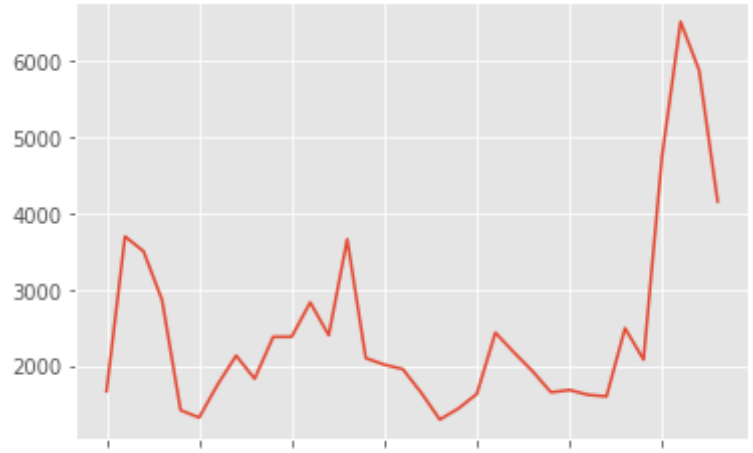
In [35]:
```
haiti = df_can.loc['Haiti', years] # passing in years 1980 - 2013 to exclude the 'total' column
haiti.head()
```

Out[35]:
```
1980    1666
1981    3692
1982    3498
1983    2860
1984    1418
Name: Haiti, dtype: object
```

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

In [36]:
```
haiti.plot()
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1171f3978>
```



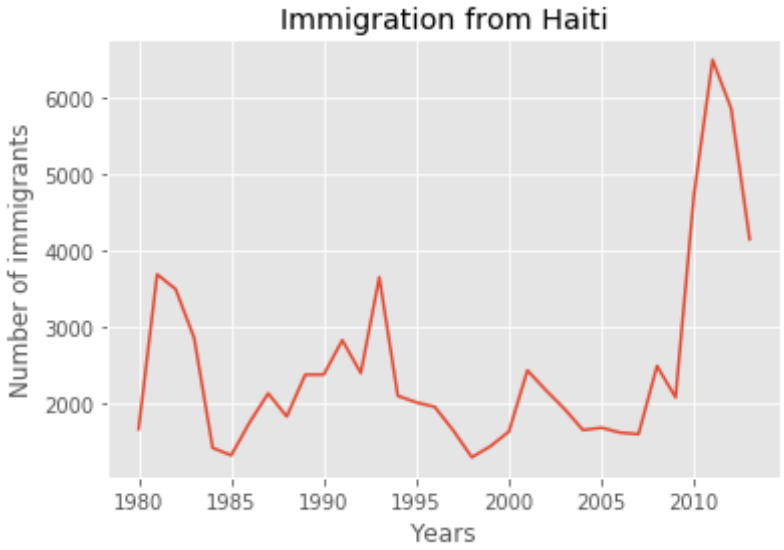
pandas automatically populated the x-axis with the index values (years), and the y-axis with the column values (population). However, notice how the years were not displayed because they are of type *string*. Therefore, let's change the type of the index values to *integer* for plotting.

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

In [37]:
```
haiti.index = haiti.index.map(int) # let's change the index values of H
aiti to type integer for plotting
haiti.plot(kind='line')

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

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



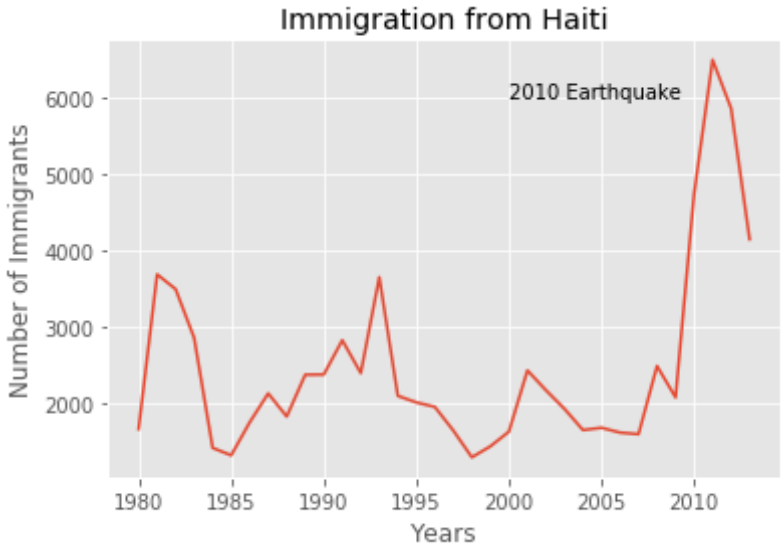
We can clearly notice how number of immigrants from Haiti spiked up from 2010 as Canada stepped up its efforts to accept refugees from Haiti. Let's annotate this spike in the plot by using the `plt.text()` method.

In [38]:
```
haiti.plot(kind='line')

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

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

plt.show()
```



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

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

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

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

```
If the years were stored as type 'string', we would need to spec
ify x as the index position of the year. Eg 20th index is year 2
000 since it is the 20th year with a base year of 1980.
```

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

```
We will cover advanced annotation methods in later modules.
```

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

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

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

In [39]:
```
### type your answer here
df_CI = df_can.loc[['China', 'India'], years]
df_CI
```

Out[39]:

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2004	2005	2006
China	5123	6682	3308	1863	1527	1816	1960	2643	2758	4323	...	36619	42584	33518
India	8880	8670	8147	7338	5704	4211	7150	10189	11522	10343	...	28235	36210	33848

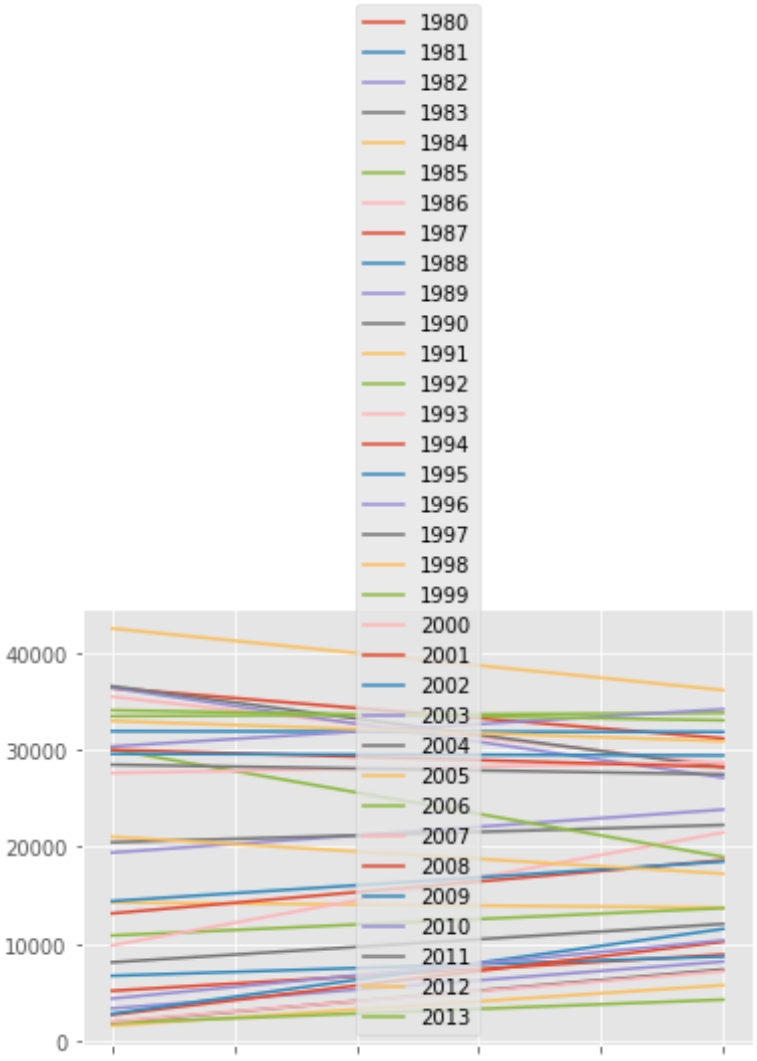
2 rows × 34 columns

Double-click **here** for the solution.

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

In [40]:
```
### type your answer here
df_CI.plot(kind='line')
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1193f24a8>
```



Double-click **here** for the solution.

That doesn't look right...

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

In [41]:
```
df_CI = df_CI.transpose()
df_CI.head()
```

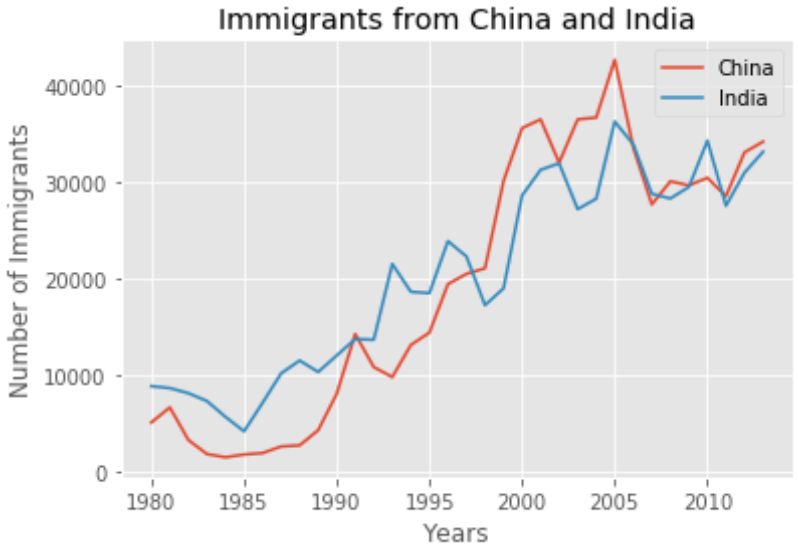
Out[41]:

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

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

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


### type your answer here
df_CI.index = df_CI.index.map(int) # let's change the index values of d
f_CI to type integer for plotting
df_CI.plot(kind='line')
plt.title('Immigrants from China and India')
plt.xlabel('Years')
plt.ylabel('Number of Immigrants')
plt.show()
```



Double-click **here** for the solution.

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

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

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

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

```
class 'pandas.core.series.Series'
1980 1666
1981 3692
1982 3498
```

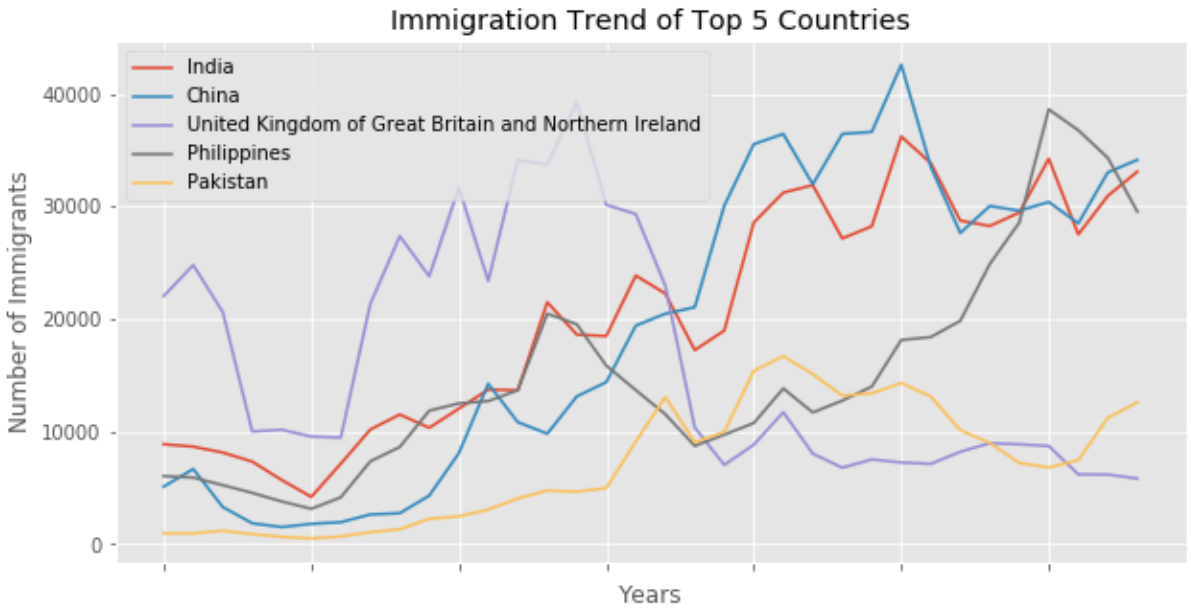
```
1983 2860
1984 1418
Name: Haiti, dtype: int64
```

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

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

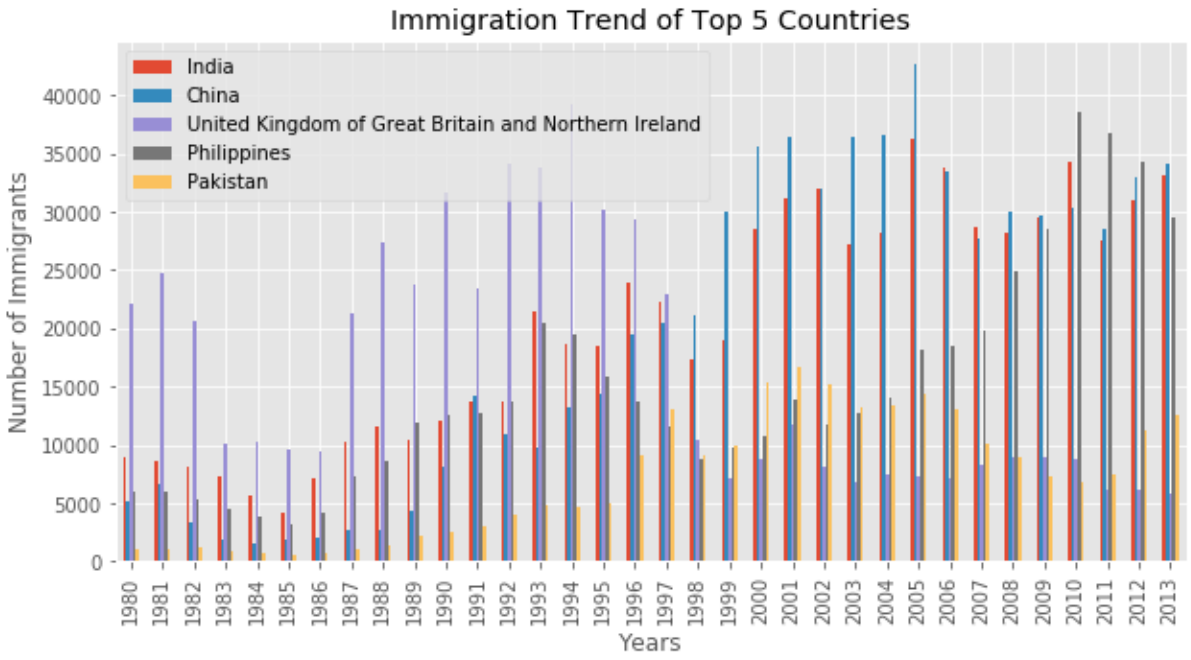
In [43]:
```
### type your answer here


df_can.sort_values(by='Total', ascending=False, inplace=True)
df_top5 = df_can.head(5)
df_top5_t = df_top5[years].transpose()
df_top5_t.plot(kind='line', figsize=(10, 5)) # pass a tuple (x, y) size
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



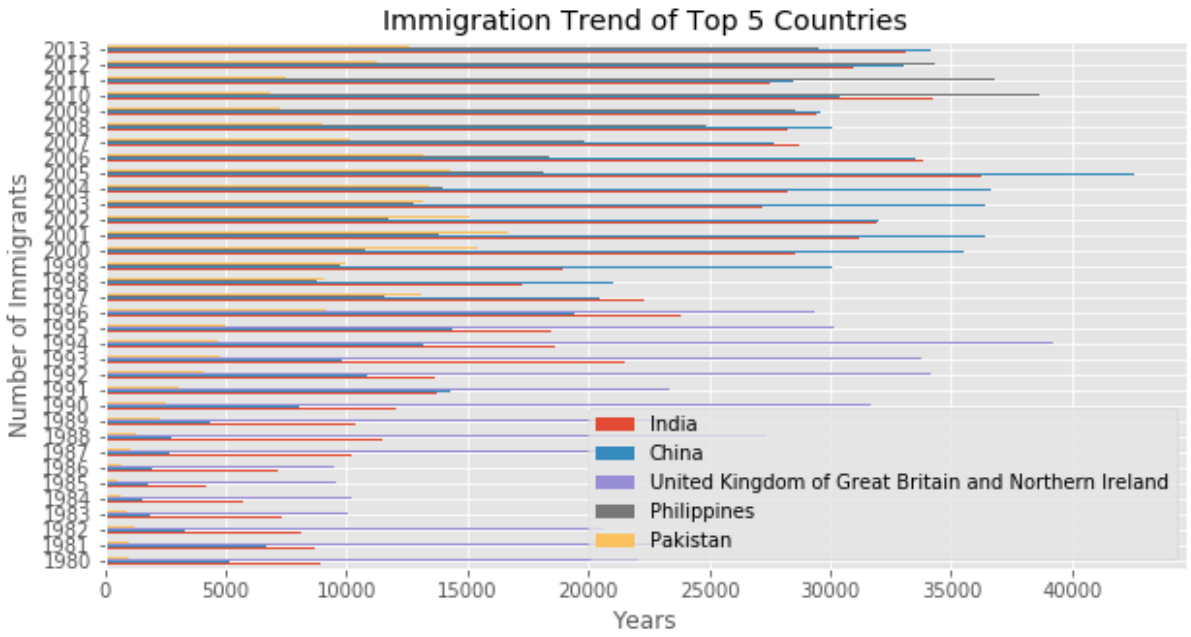
Using bar for vertical bar plots

In [44]:
```
df_can.sort_values(by='Total', ascending=False, inplace=True)
df_top5 = df_can.head(5)
df_top5_t = df_top5[years].transpose()
df_top5_t.plot(kind='bar', figsize=(10, 5)) # pass a tuple (x, y) size
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



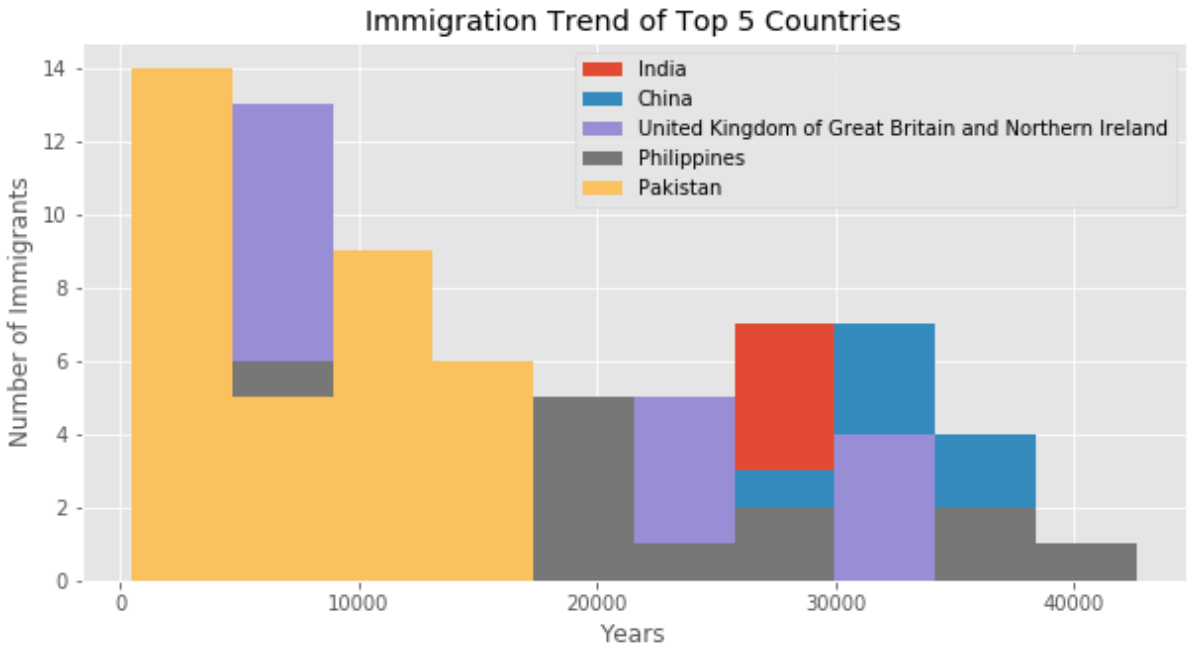
Using barh for horizontal bar plots

In [45]:
```
df_can.sort_values(by='Total', ascending=False, inplace=True)
df_top5 = df_can.head(5)
df_top5_t = df_top5[years].transpose()
df_top5_t.plot(kind='barh', figsize=(10, 5)) # pass a tuple (x, y) size
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



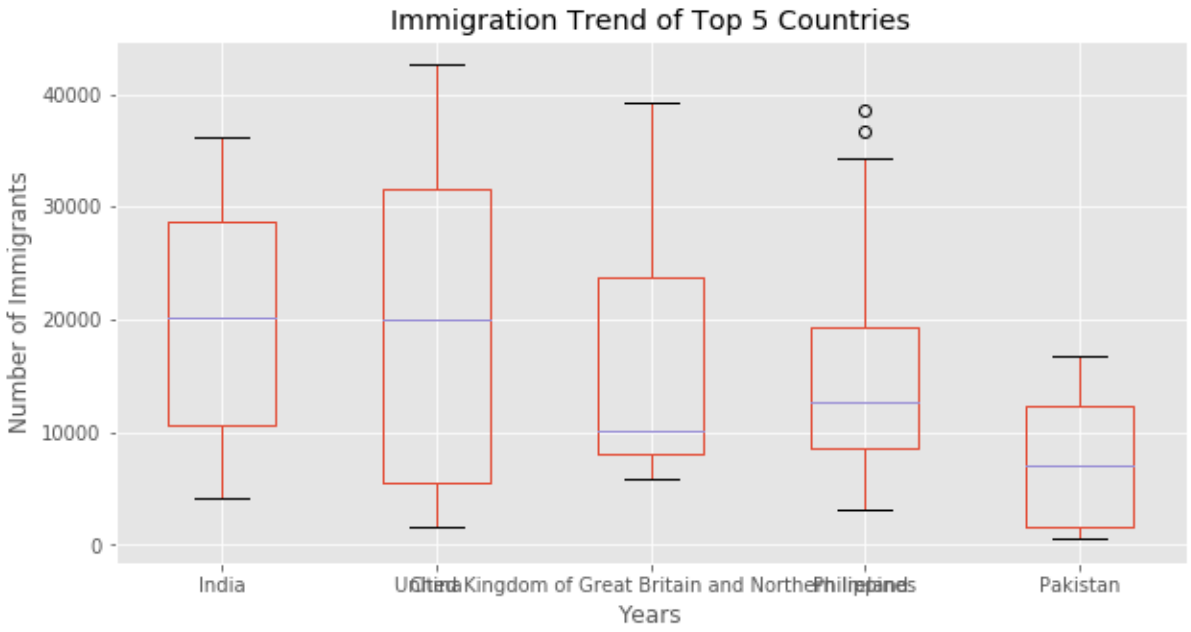
Using hist for histogram

In [46]:
```
df_can.sort_values(by='Total', ascending=False, inplace=True)
df_top5 = df_can.head(5)
df_top5_t = df_top5[years].transpose()
df_top5_t.plot(kind='hist', figsize=(10, 5)) # pass a tuple (x, y) size
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



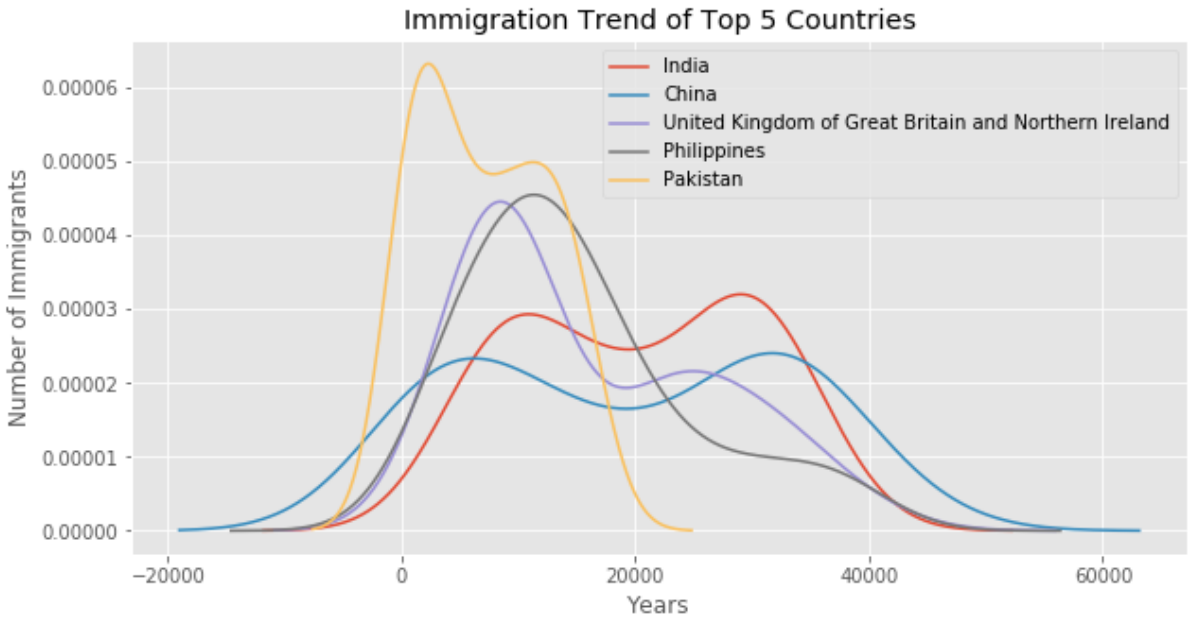
Using box for boxplots

In [47]:
```
df_can.sort_values(by='Total', ascending=False, inplace=True)
df_top5 = df_can.head(5)
df_top5_t = df_top5[years].transpose()
df_top5_t.plot(kind='box', figsize=(10, 5)) # pass a tuple (x, y) size
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



Using kde or density for density plot

In [48]:
```
df_can.sort_values(by='Total', ascending=False, inplace=True)
df_top5 = df_can.head(5)
df_top5_t = df_top5[years].transpose()
df_top5_t.plot(kind='kde', figsize=(10, 5)) # pass a tuple (x, y) size
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```



Area plots

In [54]:
```
df_can.sort_values(by='Total', ascending=False, inplace=True)
df_top5 = df_can.head(5)
df_top5_t = df_top5[years].transpose()
df_top5_t.plot(kind='area', figsize=(10, 5)) # pass a tuple (x, y) size
plt.title('Immigration Trend of Top 5 Countries')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```