Pie Charts, Box Plots, Scatter Plots, and Bubble Plots

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Explore Matplotlib library further
- Create pie charts, box plots, scatter plots and bubble charts

Table of Contents

Exploring Datasets with *pandas* and Matplotlib

Toolkits: The course heavily relies on *pandas* and **Numpy** for data wrangling, analysis, and visualization. The primary plotting library we will explore in the course is Matplotlib.

Dataset: Immigration to Canada from 1980 to 2013 - International migration flows to and from selected countries - The 2015 revision from United Nation's website.

The dataset contains annual data on the flows of international migrants as recorded by the countries of destination. The data presents both inflows and outflows according to the place of birth, citizenship or place of previous / next residence both for foreigners and nationals. In this lab, we will focus on the Canadian Immigration data.

Downloading and Prepping Data

Import primary modules.

In [2]:
```
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
```

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

```
! pip3 install openpyxl
```

Download the dataset and read it into a *pandas* dataframe.

In [3]:
```
df_can = pd.read_excel(
    'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-Skil
    sheet_name='Canada by Citizenship',
    skiprows=range(20),
    skipfooter=2
    )

print('Data downloaded and read into a dataframe!')
```

Data downloaded and read into a dataframe!

Let's take a look at the first five items in our dataset.

In [4]:
```
df_can.head()
```

Out[4]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2004	2005	2006	2007	2008	2009
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	...	2978	3436	3009	2652	2111	1746
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	...	1450	1223	856	702	560	716
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	...	3616	3626	4807	3623	4005	5393
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	...	0	0	1	0	0	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	...	0	0	1	1	0	0

5 rows × 43 columns

Let's find out how many entries there are in our dataset.

In [5]:
```
# print the dimensions of the dataframe
print(df_can.shape)
```

(195, 43)

Clean up data. We will make some modifications to the original dataset to make it easier to create our visualizations. Refer to *Introduction to Matplotlib and Line Plots* and *Area Plots, Histograms, and Bar Plots* for a detailed description of this preprocessing.

In [6]:
```
# clean up the dataset to remove unnecessary columns (eg. REG)
df_can.drop(['AREA', 'REG', 'DEV', 'Type', 'Coverage'], axis=1, inplace=True)

# let's rename the columns so that they make sense
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent','RegName':'Region'}, inplace=True)

# for sake of consistency, let's also make all column labels of type string
df_can.columns = list(map(str, df_can.columns))

# set the country name as index - useful for quickly looking up countries using .loc method
df_can.set_index('Country', inplace=True)

# add total column
df_can['Total'] = df_can.sum(axis=1)

# years that we will be using in this lesson - useful for plotting later on
years = list(map(str, range(1980, 2014)))
print('data dimensions:', df_can.shape)
```

data dimensions: (195, 38)

Visualizing Data using Matplotlib

Import `Matplotlib`.

In [7]:
```
%matplotlib inline

import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.style.use('ggplot') # optional: for ggplot-like style

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

Matplotlib version: 3.3.4

Pie Charts

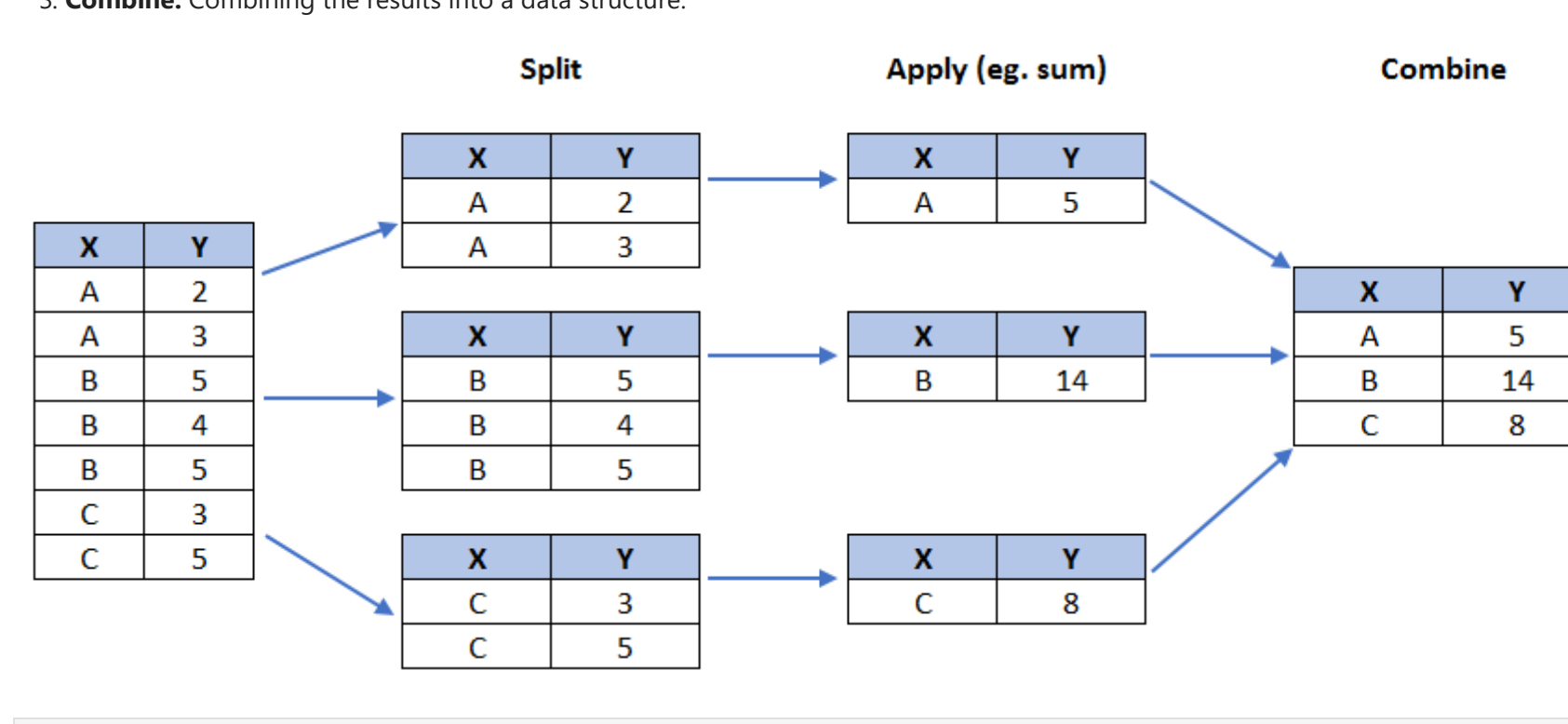
A `pie chart` is a circular graphic that displays numeric proportions by dividing a circle (or pie) into proportional slices. You are most likely already familiar with pie charts as it is widely used in business and media. We can create pie charts in Matplotlib by passing in the `kind=pie` keyword.

Let's use a pie chart to explore the proportion (percentage) of new immigrants grouped by continents for the entire time period from 1980 to 2013.

Step 1: Gather data.

We will use *pandas* `groupby` method to summarize the immigration data by `Continent`. The general process of `groupby` involves the following steps:

1. **Split:** Splitting the data into groups based on some criteria.
2. **Apply:** Applying a function to each group independently: .sum() .count() .mean() .std() .aggregate() .apply() .etc..
3. **Combine:** Combining the results into a data structure.



In [8]:
```
# group countries by continents and apply sum() function
df_continents = df_can.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_can.groupby('Continent', axis=0)))

df_continents.head()
```

pandas.core.groupby.generic.DataFrameGroupBy

Out[8]:

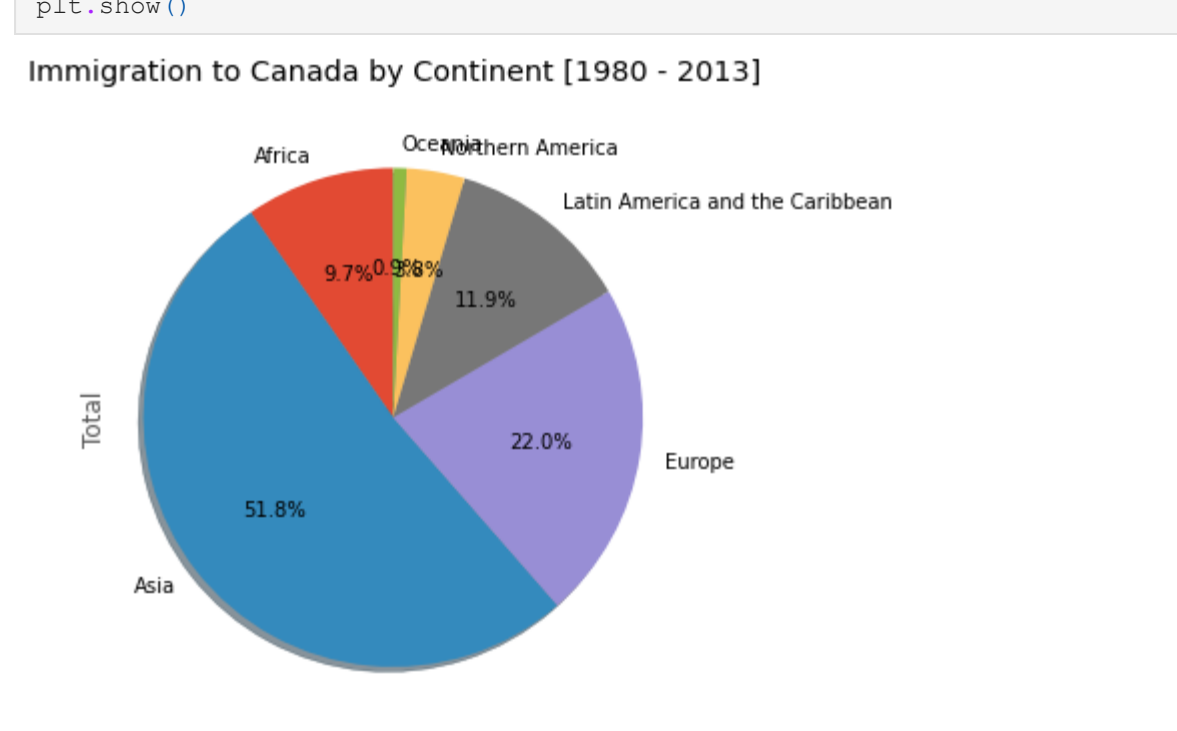
	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010
Continent																	
Africa	3951	4363	3819	2671	2639	2650	3782	7494	7552	9894	...	27523	29188	28284	29890	34534	40892
Asia	31025	34314	30214	24696	27274	23850	28739	43203	47454	60256	...	159253	149054	133459	139894	141434	163845
Europe	39760	44802	42720	24638	22287	20844	24370	46698	54726	60893	...	35955	33053	33495	34692	35078	33425
Latin America and the Caribbean	13081	15215	16769	15427	13678	15171	21179	28471	21924	25060	...	24747	24676	26011	26547	26867	28818
Northern America	9378	10030	9074	7100	6661	6543	7074	7705	6469	6790	...	8394	9613	9463	10190	8995	8142

5 rows × 35 columns

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).

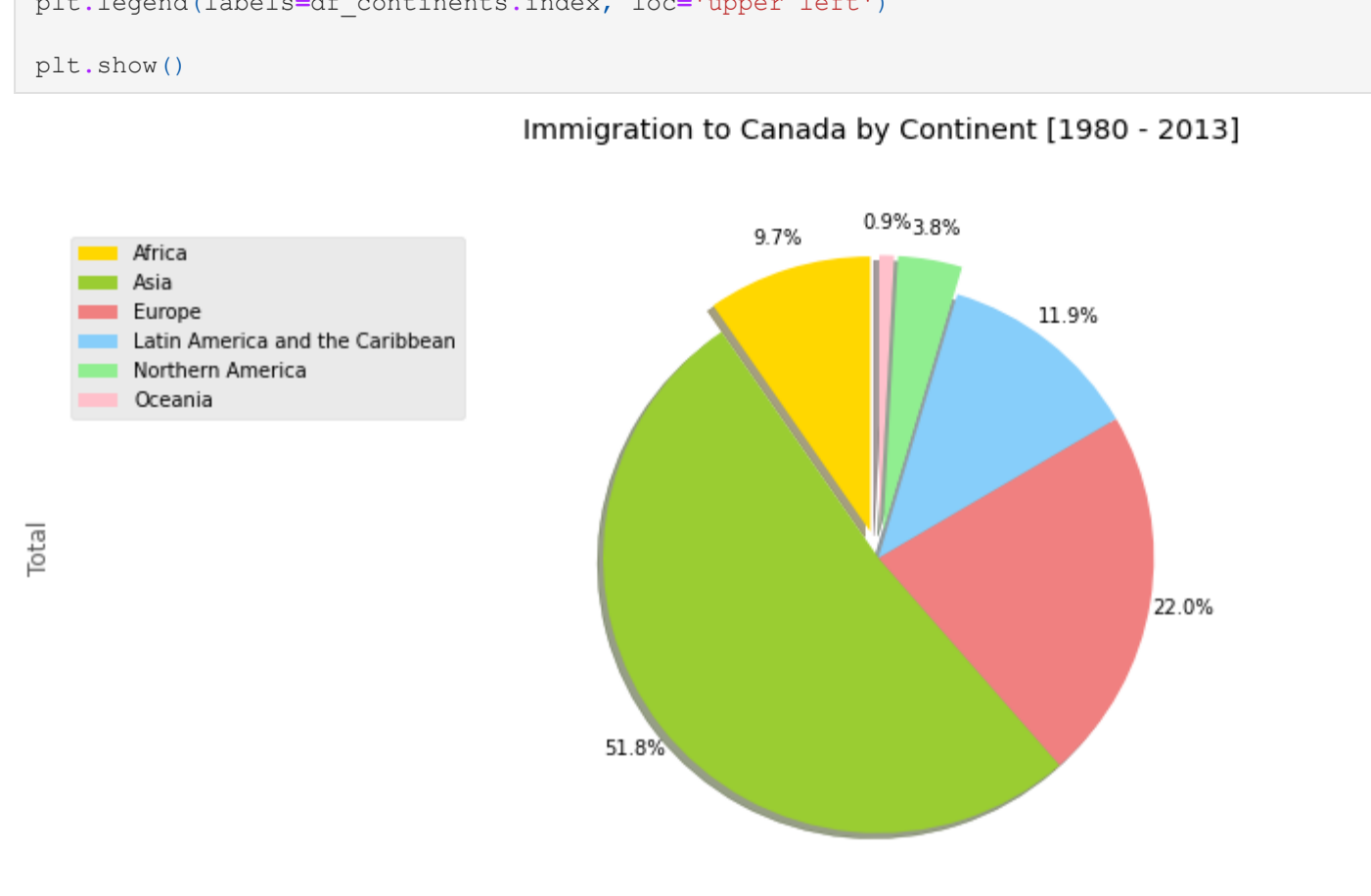
In [9]:
```
# autopct create %, start angle represent starting point
df_continents['Total'].plot(kind='pie',
                            figsize=(5, 6),
                            autopct='%1.1f%%', # add in percentages
                            startangle=90,     # start angle 90° (Africa)
                            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()
```



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 Caribbean) by passing in `explode` parameter.

In [10]:
```
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.

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
                            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()
```

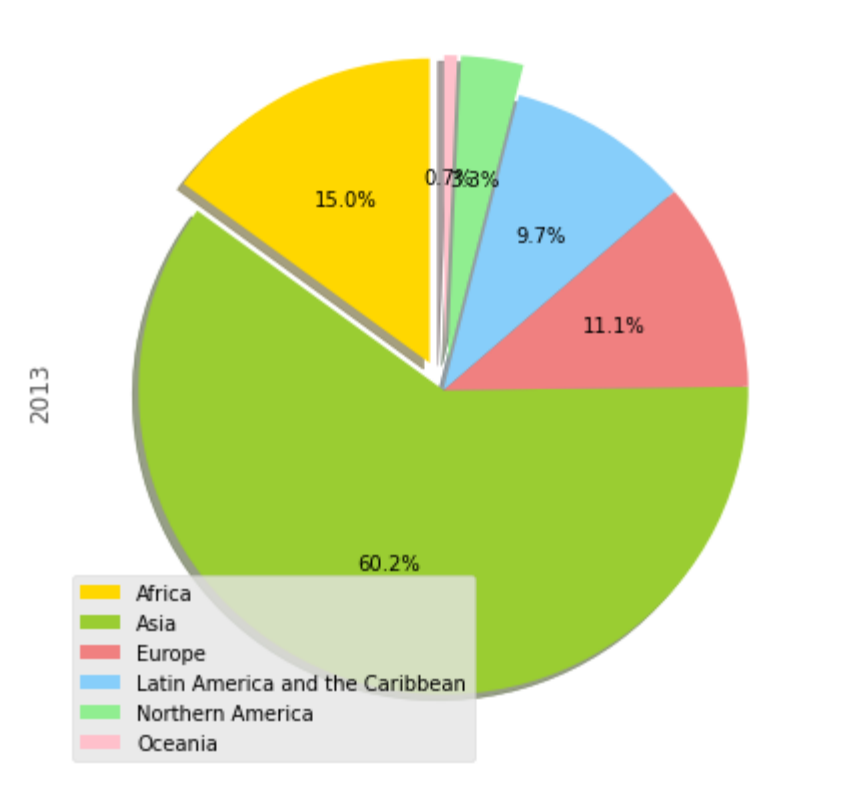


Question: Using a pie chart, explore the proportion (percentage) of new immigrants grouped by continents in the year 2013.

Note: You might need to play with the explore values in order to fix any overlapping slice values.

In [11]:
```
### type your answer here

newdf= df_can.groupby('Continent').sum()

newdf['2013'].plot(kind='pie',figsize=(16,7),autopct='%1.1f%%',startangle=90,shadow=True,colors=colors_list,lab

plt.legend(newdf.index,loc='best')
```

Out[11]: <matplotlib.legend.Legend at 0x7f97824a6710>

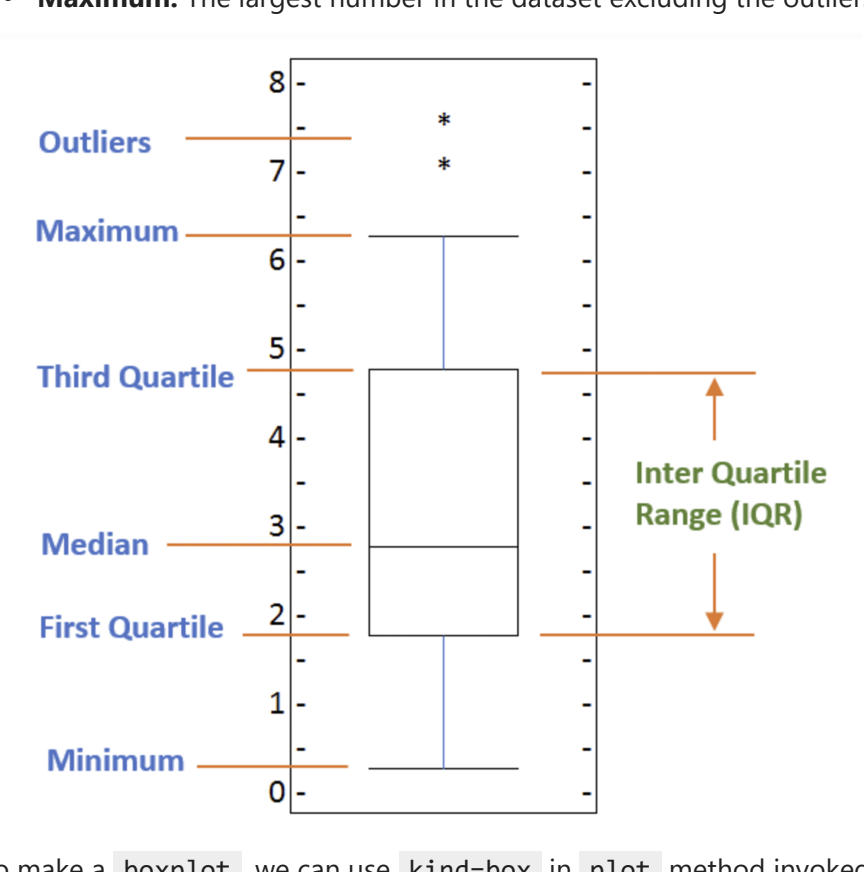


▶ Click here for a sample python solution

Box Plots

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

- **Minimum:** The smallest number in the dataset excluding the outliers.
- **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:** The largest number in the dataset excluding the outliers.



To make a `boxplot`, we can use `kind=box` in `plot` method invoked on a *pandas* series or dataframe.

Let's plot the box plot for the Japanese immigrants between 1980 - 2013.

In [78]:
```
df_can.head()
```

Out[78]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011
Country																		
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223	856	702	560	716	561	539
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626	4807	3623	4005	5393	4752	4325
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0	...	0	1	0	0	0	0	0
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2	...	0	1	1	0	0	0	0

5 rows × 38 columns

Step 1: Get the subset of the dataset. Even though we are extracting the data for just one country, we will obtain it as a dataframe. This will help us with calling the `dataframe.describe()` method to view the percentiles.

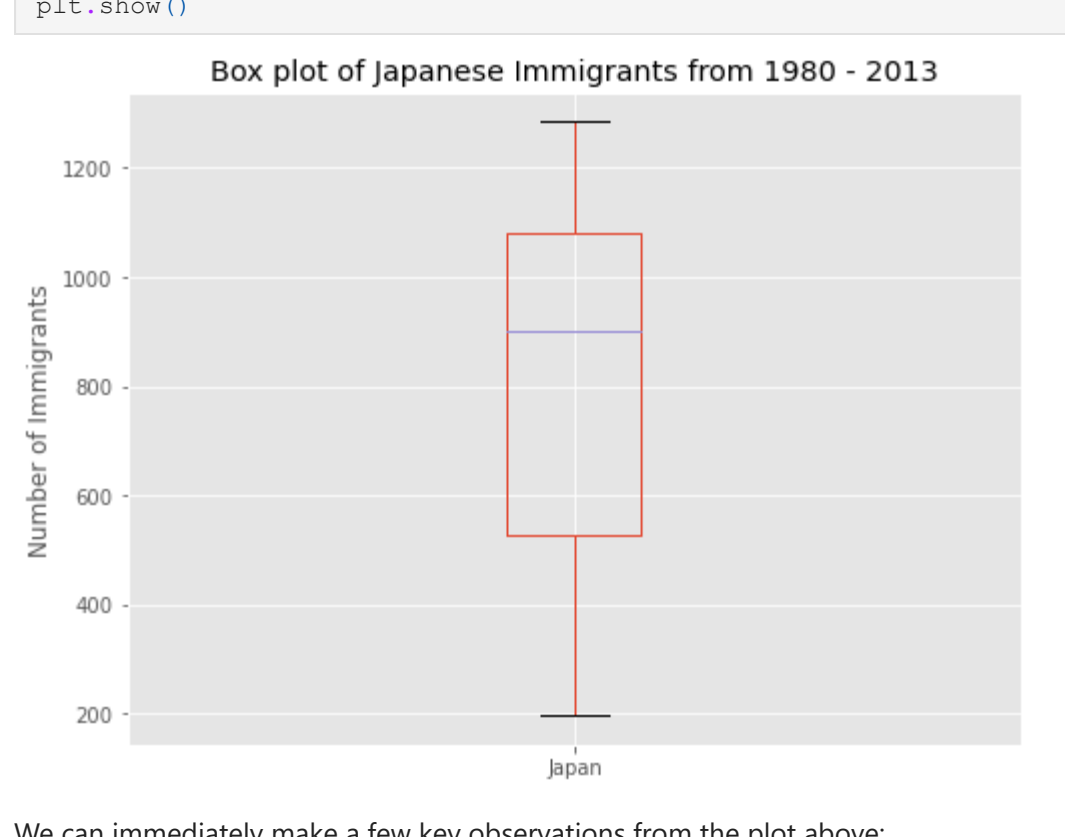
In [12]:
```
# to get a dataframe, place extra square brackets around 'Japan'.
df_japan = df_can.loc[['Japan'], years].transpose()
df_japan.head()
```

Out[12]:

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

Step 2: Plot by passing in `kind='box'`.

In [13]:
```
df_japan.plot(kind='box', figsize=(8, 6))

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

plt.show()
```



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

1. The minimum number of immigrants is around 200 (min), maximum number is around 1300 (max), and median number of immigrants is around 900 (median).
2. 25% of the years for period 1980 - 2013 had an annual immigrant count of ~500 or fewer (First quartile).
3. 75% of the years for period 1980 - 2013 had an annual immigrant count of ~1100 or fewer (Third quartile).

We can view the actual numbers by calling the `describe()` method on the dataframe.

In [14]:
```
df_japan.describe()
```

Out[14]:

Country	Japan
count	34.000000
mean	814.911765
std	337.219771
min	198.000000
25%	529.000000
50%	902.000000
75%	1079.000000
max	1284.000000

One of the key benefits of box plots is comparing the distribution of multiple datasets. In one of the previous labs, we observed that China and India had very similar immigration trends. Let's analyze these two countries further using box plots.

Question: Compare the distribution of the number of new immigrants from India and China for the period 1980 - 2013.

Step 1: Get the dataset for China and India and call the dataframe **df_CI**.

In [29]:
```
### type your answer here
df_ci = df_can.loc[['China','India']]
df_ci.shape
```

Out[29]: (2, 38)

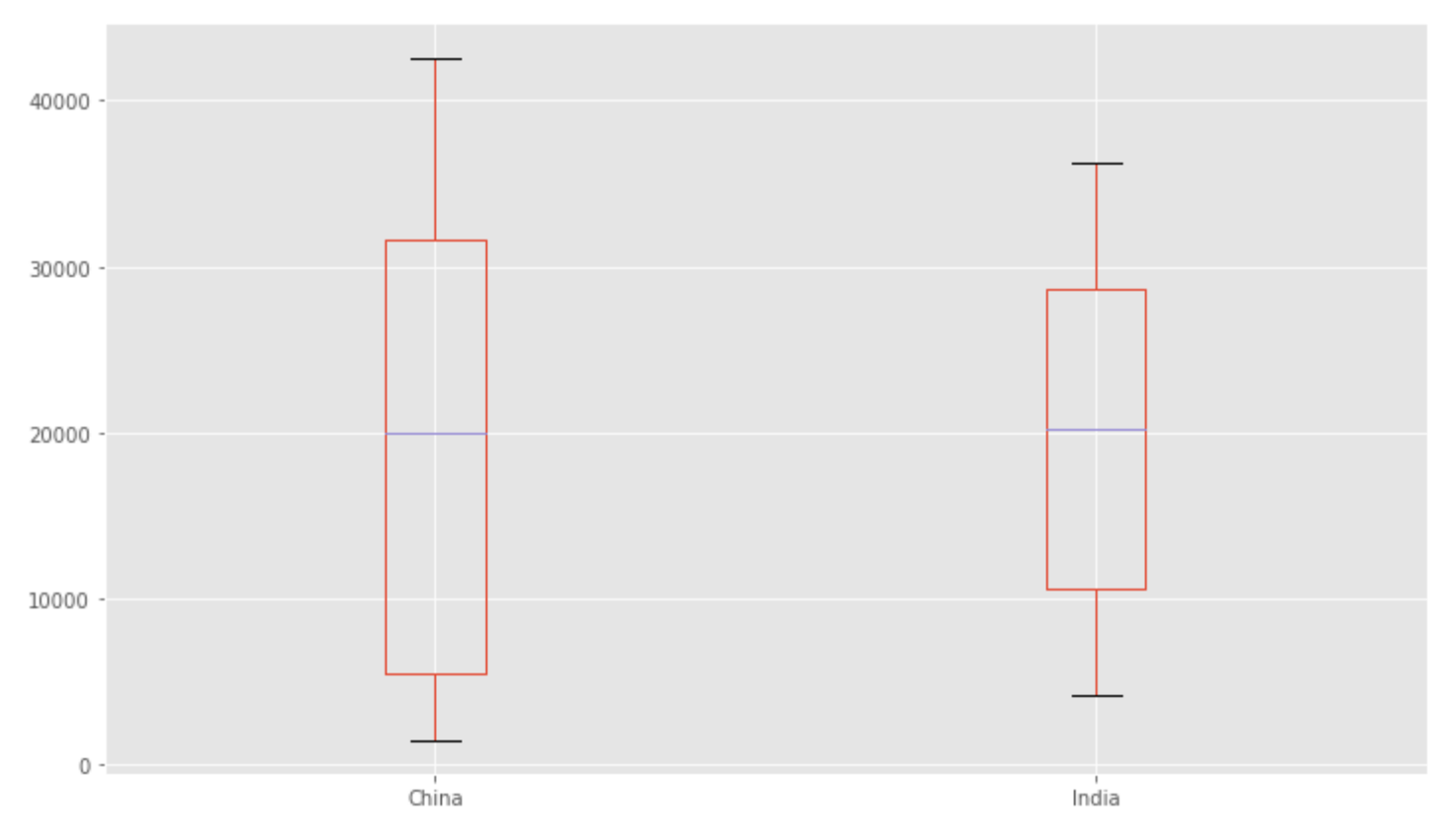
In [31]:
```
df_ci.iloc[0:1,3:37].transpose()
```

Out[31]:

Country	China
1980	5123
1981	6682
1982	3308
1983	1863
1984	1527
1985	1816
1986	1960
1987	2643
1988	2758
1989	4323
1990	8076
1991	14255
1992	10846
1993	9817
1994	13128
1995	14398
1996	19415
1997	20475
1998	21049
1999	30069
2000	35529
2001	36434
2002	31961
2003	36439
2004	36619
2005	42584
2006	33518
2007	27642
2008	30037
2009	29622
2010	30391
2011	28502
2012	33024
2013	34129

In [36]:
```
df_can.loc[['China','India'], years].transpose().plot(kind='box',figsize=(12,7))
```

Out[36]: <AxesSubplot:>



▶ Click here for a sample python solution

Let's view the percentiles associated with both countries using the `describe()` method.

In [37]:
```
### type your answer here

df_ci.describe()
```

Out[37]:

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...
count	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.00000	2.000000	...
mean	7001.500000	7676.000000	5727.500000	4600.500000	3615.500000	3013.500000	4555.000000	6416.000000	7140.00000	7333.000000	...
std	2656.600177	1405.728281	3421.689714	3871.409627	2953.585025	1693.520741	3669.884194	5335.827771	6197.08383	4256.782823	...
min	5123.000000	6682.000000	3308.000000	1863.000000	1527.000000	1816.000000	1960.000000	2643.000000	2758.00000	4323.000000	...
25%	6062.250000	7179.000000	4517.750000	3231.750000	2571.250000	2414.750000	3257.500000	4529.500000	4949.00000	5828.000000	...
50%	7001.500000	7676.000000	5727.500000	4600.500000	3615.500000	3013.500000	4555.000000	6416.000000	7140.00000	7333.000000	...
75%	7940.750000	8173.000000	6937.250000	5969.250000	4659.750000	3612.250000	5852.500000	8302.500000	9331.00000	8838.000000	...
max	8880.000000	8670.000000	8147.000000	7338.000000	5704.000000	4211.000000	7150.000000	10189.000000	11522.00000	10343.000000	...

8 rows × 35 columns

In [43]:
```
df_ci.transpose().head()
```

Out[43]:

Country	China	India
Continent	Asia	Asia
Region	Eastern Asia	Southern Asia
DevName	Developing regions	Developing regions
1980	5123	8880
1981	6682	8670

▶ Click here for a sample python solution

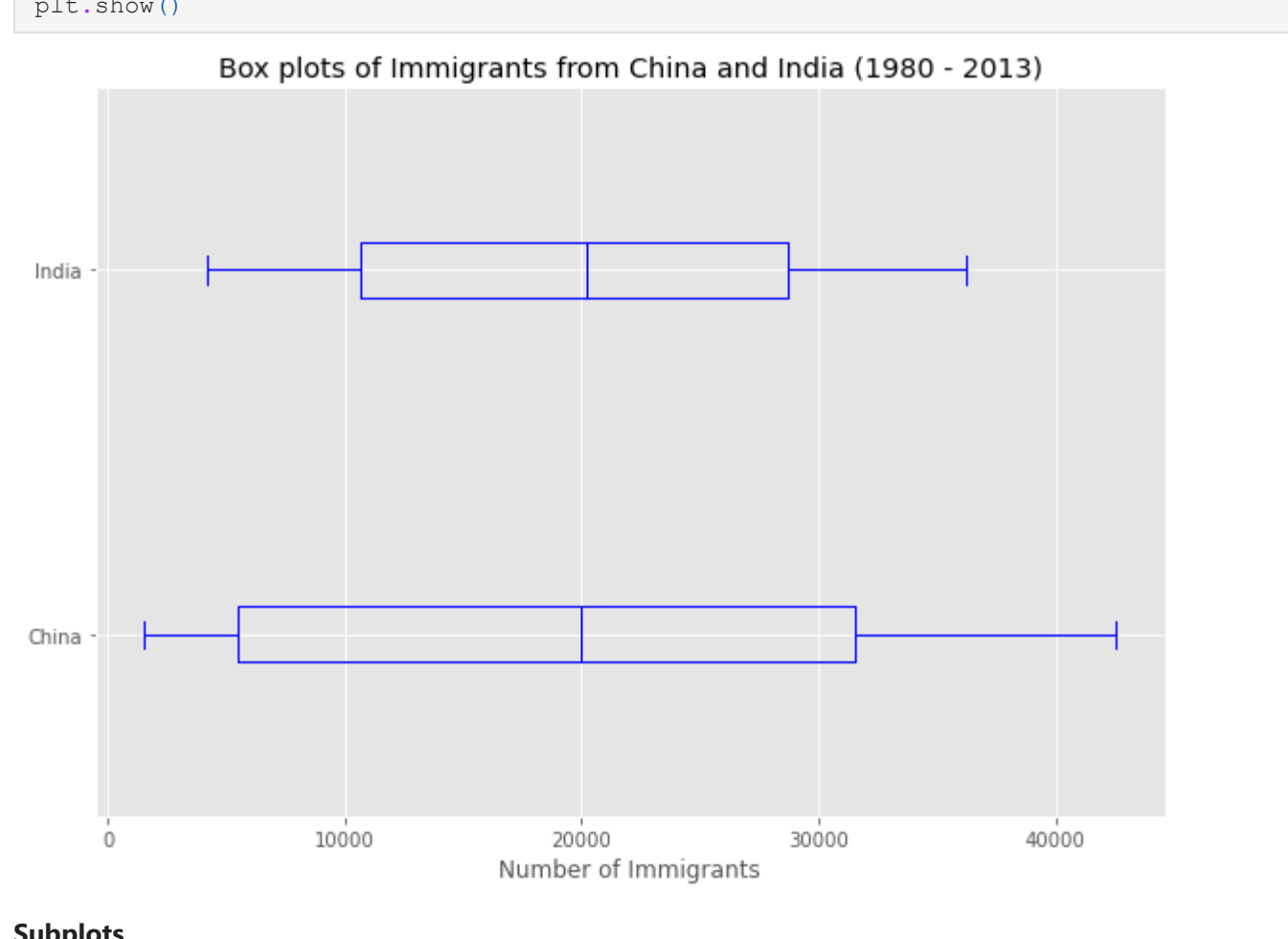
Step 2: Plot data.

▶ Click here for a sample python solution

We can observe that, while both countries have around the same median immigrant population (~20,000), China's immigrant population range is more spread out than India's. The maximum population from India for any year (36,210) is around 15% lower than the maximum population from China (42,584).

If you prefer to create horizontal box plots, you can pass the `vert` parameter in the **plot** function and assign it to *False*. You can also specify a different color in case you are not a big fan of the default red color.

In [45]:
```
# horizontal box plots
df_ci.transpose().iloc[3:37,].plot(kind='box', figsize=(10, 7), color='blue', vert=False)

plt.title('Box plots of Immigrants from China and India (1980 - 2013)')
plt.xlabel('Number of Immigrants')

plt.show()
```



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.

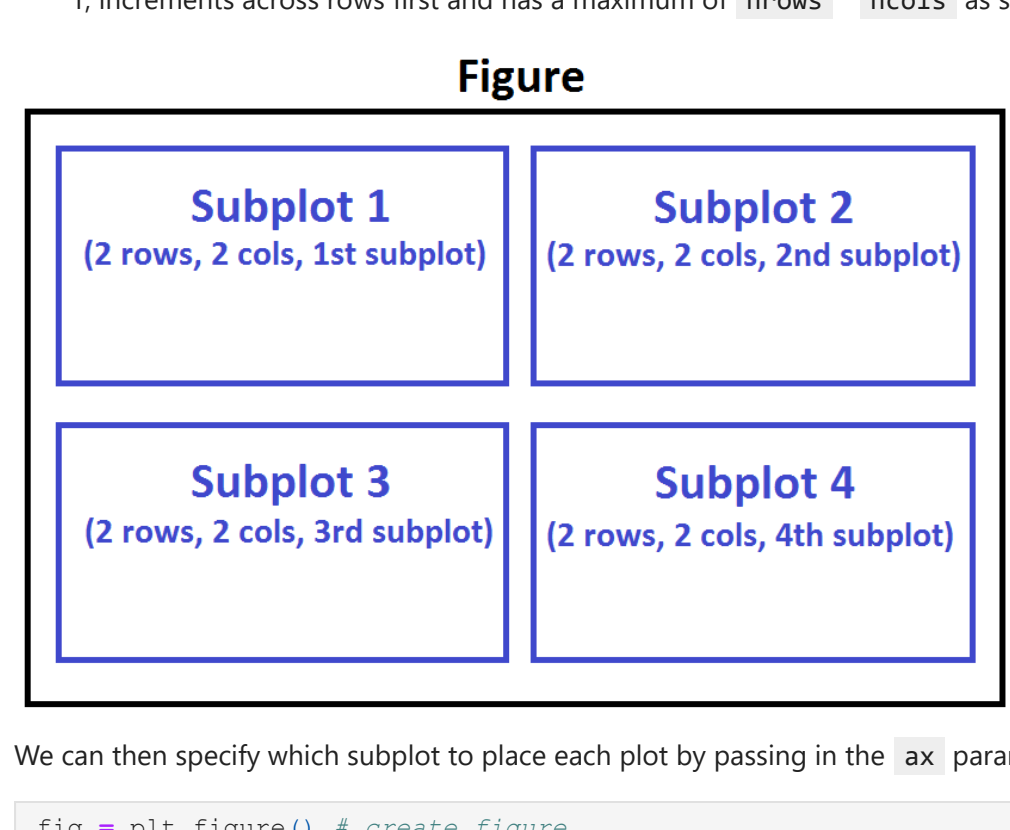
To visualize multiple plots together, we can create a **`figure`** (overall canvas) and divide it into **`subplots`**, each containing a plot. With **subplots**, we usually work with the **artist layer** instead of the **scripting layer**.

Typical syntax is :

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

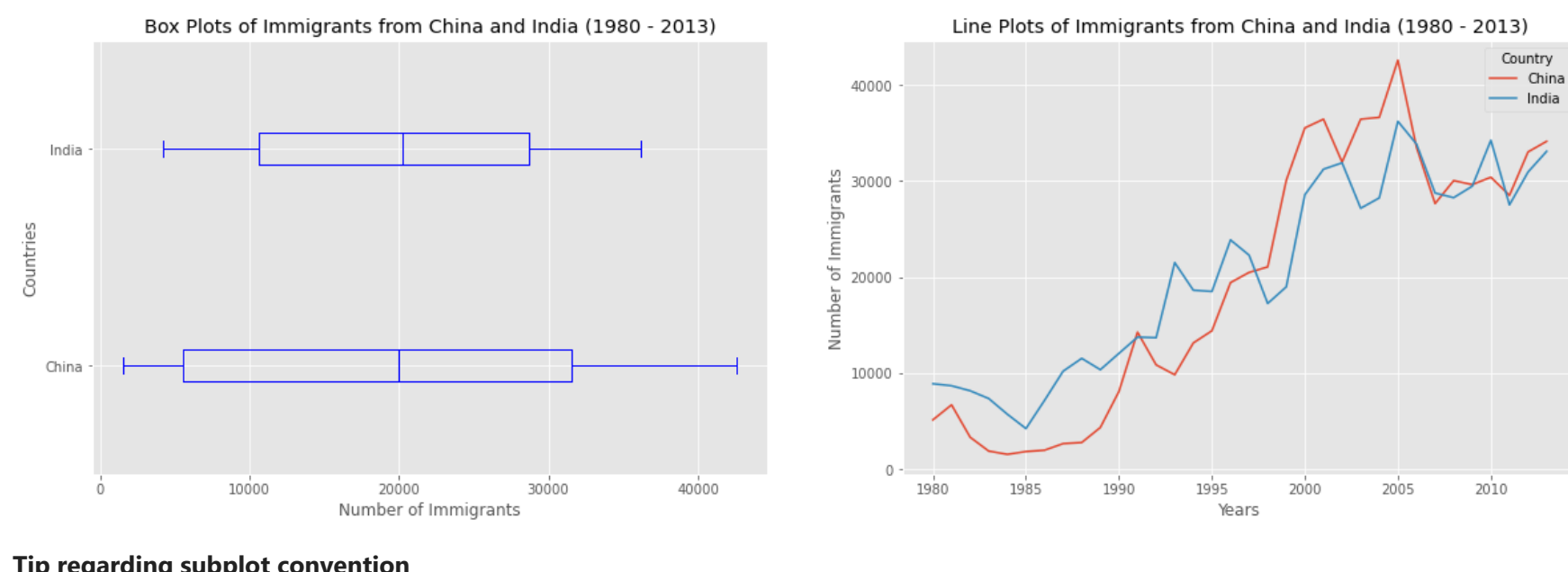
Where

- `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.



We can then specify which subplot to place each plot by passing in the `ax` paramemter in `plot()` method as follows:

In [47]:
```
fig = plt.figure() # create figure

ax0 = fig.add_subplot(1, 2, 1) # add subplot 1 (1 row, 2 columns, first plot)
ax1 = fig.add_subplot(1, 2, 2) # add subplot 2 (1 row, 2 columns, second plot). See tip below**

# Subplot 1: Box plot
df_ci.transpose().iloc[3:37,].plot(kind='box', color='blue', vert=False, figsize=(20, 6), ax=ax0) # add to subp
ax0.set_title('Box Plots of Immigrants from China and India (1980 - 2013)')
ax0.set_xlabel('Number of Immigrants')
ax0.set_ylabel('Countries')

# Subplot 2: Line plot
df_ci.transpose().iloc[3:37,].plot(kind='line', figsize=(20, 6), ax=ax1) # add to subplot 2
ax1.set_title ('Line Plots of Immigrants from China and India (1980 - 2013)')
ax1.set_ylabel('Number of Immigrants')
ax1.set_xlabel('Years')

plt.show()
```



Tip regarding subplot convention

In the case when `nrows`, `ncols`, and `plot_number` are all less than 10, a convenience exists such that a 3-digit number can be given instead, where the hundreds represent `nrows`, the tens represent `ncols` and the units represent `plot_number`. For instance,

```
subplot(211) == subplot(2, 1, 1)
```

produces a subaxes in a figure which represents the top plot (i.e. the first) in a 2 rows by 1 column notional grid (no grid actually exists, but conceptually this is how the returned subplot has been positioned).

Let's try something a little more advanced.

Previously we identified the top 15 countries based on total immigration from 1980 - 2013.

Question: Create a box plot to visualize the distribution of the top 15 countries (based on total immigration) grouped by the *decades* `1980s`, `1990s`, and `2000s`.

Step 1: Get the dataset. Get the top 15 countries based on Total immigrant population. Name the dataframe **df_top15**.

In [48]:
```
### type your answer here

# df_top15 = ...
# df_top15
df_top15 = df_can.sort_values(['Total'], ascending=False, axis=0).head(15)
df_top15
```