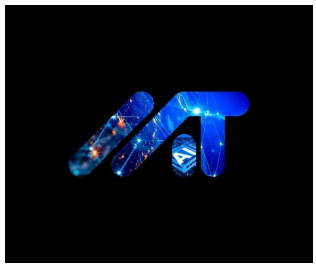


Pie Charts, Box Plots, Scatter Plots, and Bubble Plots

Importing required libraries

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Loading data

Note: All steps that are performed below are explain in detail in Tutorial

In [2]:
```
df = 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 read into a pandas dataframe!')
```

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

In [3]:
```
# in pandas axis=0 represents rows (default) and axis=1 represents columns.
df.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=True)
df.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'RegName':'Region'}, inplace=True)
df['Total'] = df.sum(axis=1)
df.set_index('Country', inplace=True)
```

```
C:\Users\Meer Moazzam\AppData\Local\Temp\ipykernel_9960\3820691460.py:4: FutureWarning: Dropping of nuisance co
lumns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Typ
eError.  Select only valid columns before calling the reduction.
  df['Total'] = df.sum(axis=1)
```

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

Out[4]:

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	201
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203	263
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223	856	702	560	716	561	539	62
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626	4807	3623	4005	5393	4752	4325	377
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

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 [5]:
```
# group countries by continents and apply sum() function
df_continents = df.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.groupby('Continent', axis=0)))

df_continents.head()
```

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

Out[5]:

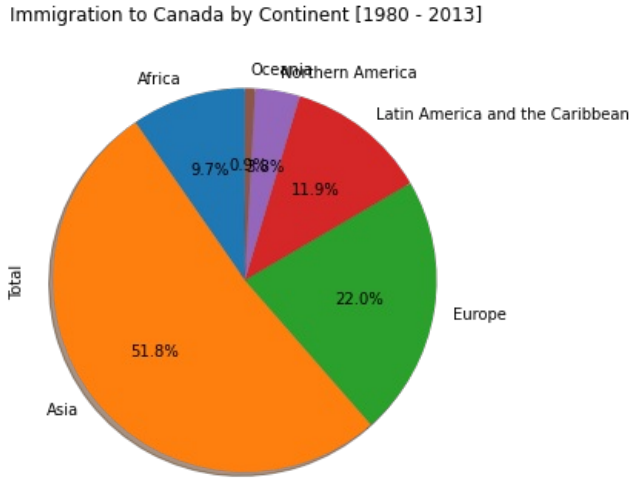
Continent	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010
Africa	3951	4363	3819	2671	2639	2650	3782	7494	7552	9894	...	27523	29188	28284	29890	34534	40892
Asia	31025	34314	30214	24696	27274	23850	28739	43203	47454	60256	...	159253	149054	133459	139894	141434	163845
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).

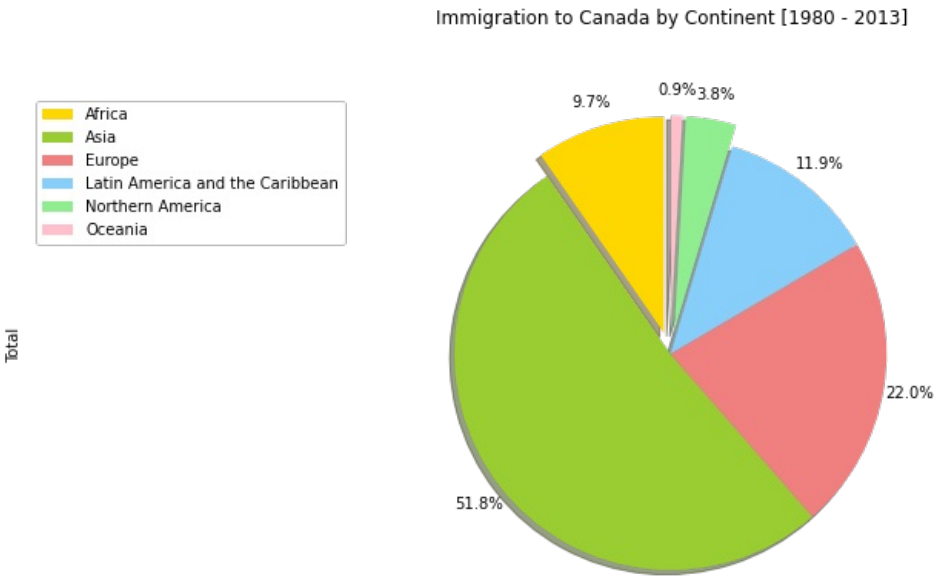
In [6]:
```
# autopct create %, start angle represent starting point
df_continents['Total'].plot(kind='pie',
                            figsize=(5, 6),
                            autopct='%1.1f%%', # add in percentages
                            startangle=90,     # start angle 90° (Africa)
                            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 [7]:
```
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.

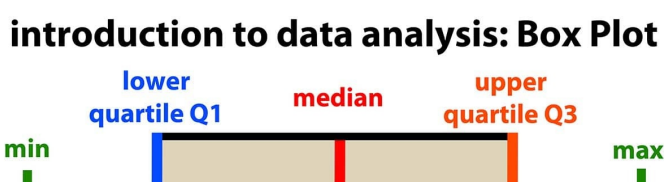
In [8]:
```
### type your answer here
```

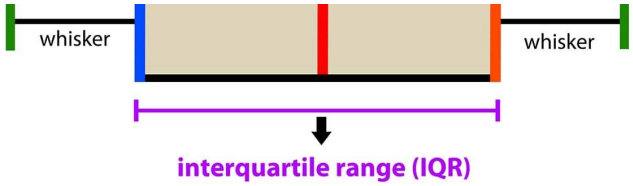
▶ Click here for a sample python solution

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.

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.

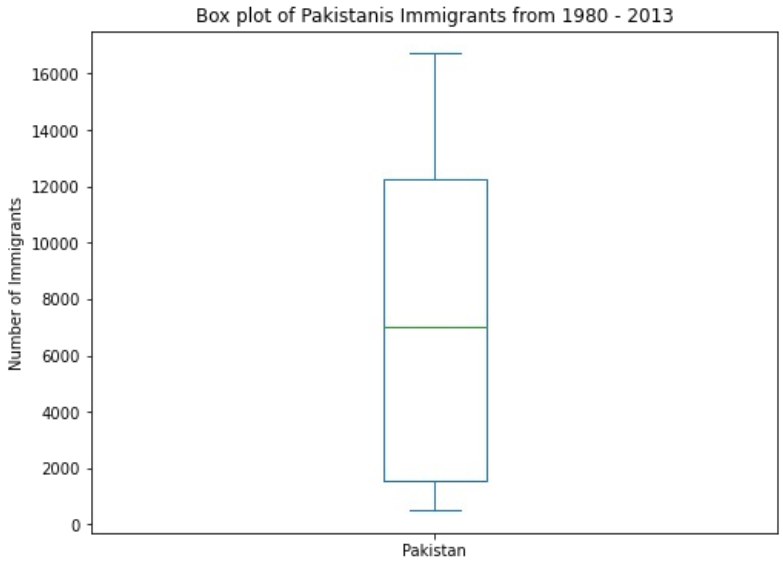
In [11]:
```
# to get a dataframe, place extra square brackets around 'Pakistan'.
years=list(range(1980,2014))
df_pak = df.loc[['Pakistan'], years].transpose()
df_pak.head()
```

Out[11]:

Country	Pakistan
1980	978
1981	972
1982	1201
1983	900
1984	668

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

In [12]:
```
df_pak.plot(kind='box', figsize=(8, 6))

plt.title('Box plot of Pakistanis 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 550 (min), maximum number is around 17000 (max), and median number of immigrants is around 7000 (median).
2. 25% of the years for period 1980 - 2013 had an annual immigrant count of ~1700 or fewer (First quartile).
3. 75% of the years for period 1980 - 2013 had an annual immigrant count of ~12000 or fewer (Third quartile).

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

In [13]:
```
df_pak.describe()
```

Out[13]:

Country	Pakistan
count	34.000000
mean	7105.882353
std	5315.849587
min	514.000000
25%	1565.750000
50%	7014.000000
75%	12259.000000
max	16708.000000

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 Pakistan and China for the period 1980 - 2013.

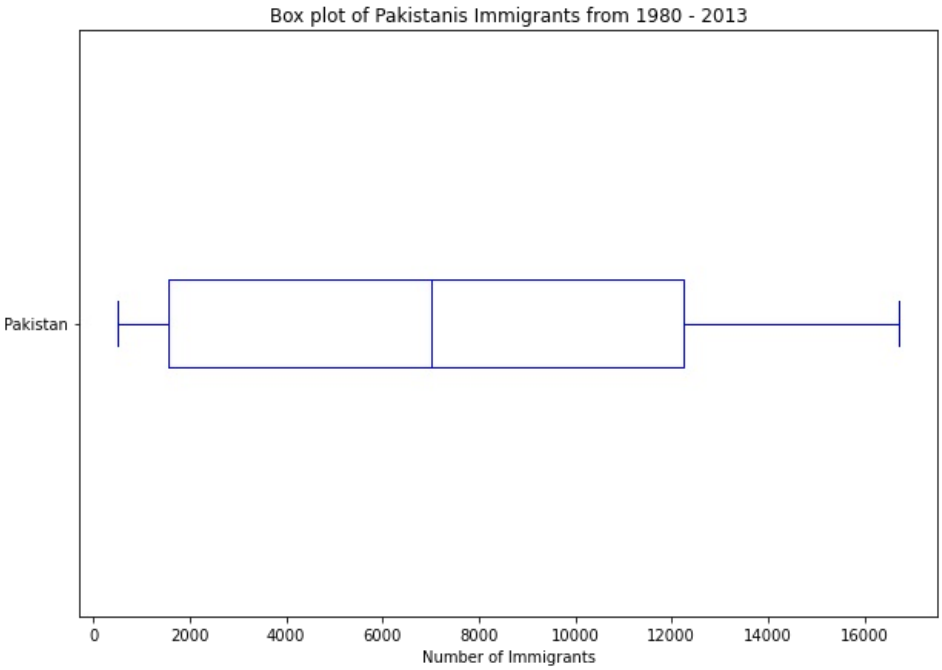
Step 1: Get the dataset for China and India and call the dataframe **df_CP**.

In [15]:
```
### type your answer here
```

▶ Click here for a sample python solution

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 [17]:
```
# horizontal box plots
df_pak.plot(kind='box', figsize=(10, 7), color='blue', vert=False)

plt.title('Box plot of Pakistanis Immigrants from 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
```

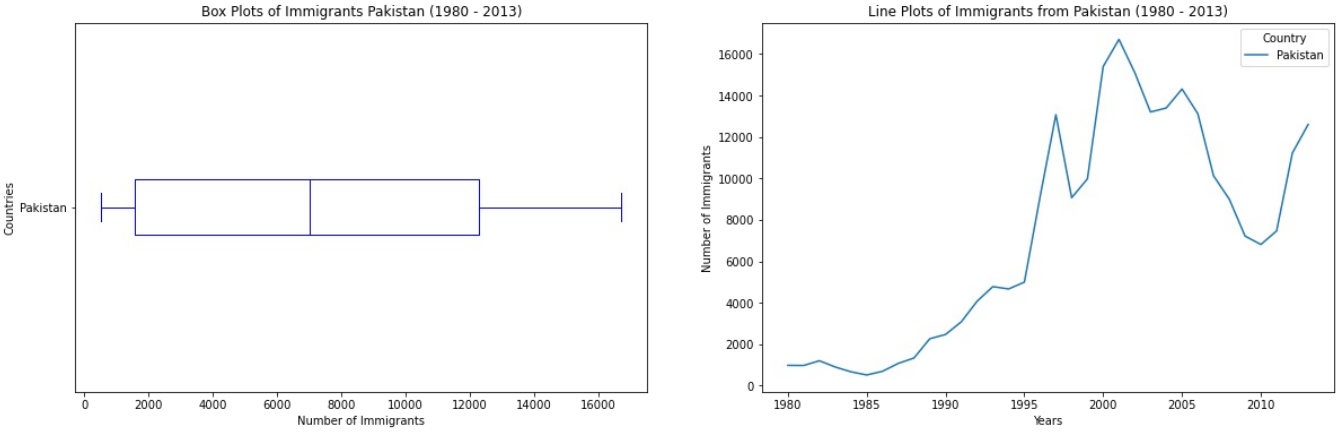
Where

- `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 [18]:
```
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_pak.plot(kind='box', color='blue', vert=False, figsize=(20, 6), ax=ax0) # add to subplot 1
ax0.set_title('Box Plots of Immigrants Pakistan (1980 - 2013)')
ax0.set_xlabel('Number of Immigrants')
ax0.set_ylabel('Countries')

# Subplot 2: Line plot
df_pak.plot(kind='line', figsize=(20, 6), ax=ax1) # add to subplot 2
ax1.set_title ('Line Plots of Immigrants from Pakistan (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`.

▶ Click here for a sample python solution

Note how the box plot differs from the summary table created. The box plot scans the data and identifies the outliers. In order to be an outlier, the data value must be:

- larger than Q3 by at least 1.5 times the interquartile range (IQR), or,
- smaller than Q1 by at least 1.5 times the IQR.

Let's look at decade 2000s as an example:

- Q1 (25%) = 36,101.5
- Q3 (75%) = 105,505.5
- IQR = Q3 - Q1 = 69,404

Using the definition of outlier, any value that is greater than Q3 by 1.5 times IQR will be flagged as outlier.

Outlier > 105,505.5 + (1.5 * 69,404)
Outlier > 209,611.5

In [19]:
```
# let's check how many entries fall above the outlier threshold
```

▶ Click here for a sample python solution

Scatter Plots

A `scatter plot` (2D) is a useful method of comparing variables against each other. `Scatter` plots look similar to `line plots` in that they both map independent and dependent variables on a 2D graph. While the data points are connected together by a line in a line plot, they are not connected in a scatter plot. The data in a scatter plot is considered to express a trend. With further analysis using tools like regression, we can mathematically calculate this relationship and use it to predict trends outside the dataset.

Let's start by exploring the following:

Using a `scatter plot`, let's visualize the trend of total immigrantion to Canada (all countries combined) for the years 1980 - 2013.

Step 1: Get the dataset. Since we are expecting to use the relationship betewen `years` and `total population`, we will convert `years` to `int` type.

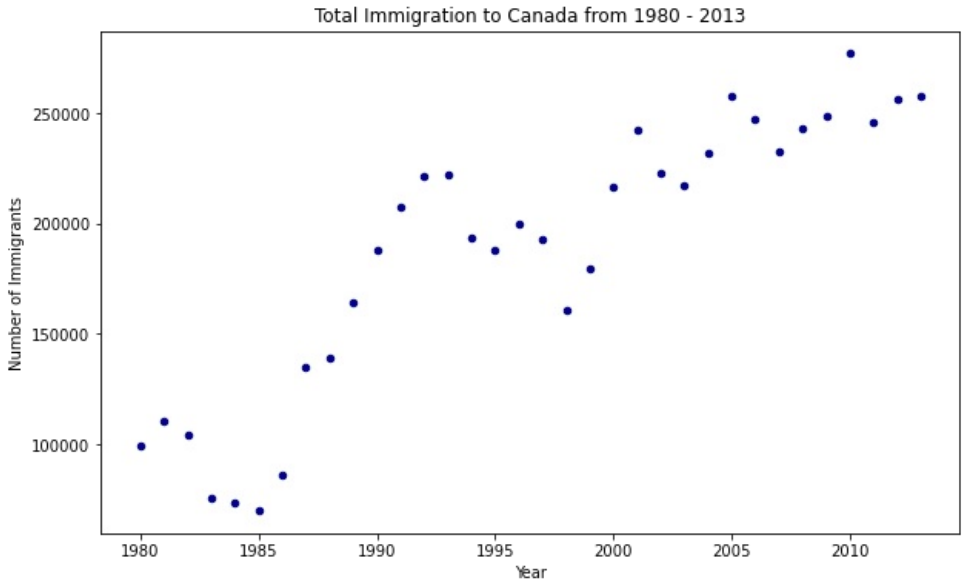
In [20]:
```
# we can use the sum() method to get the total population per year
df_tot = pd.DataFrame(df[years].sum(axis=0))

# change the years to type int (useful for regression later on)
df_tot.index = map(int, df_tot.index)

# reset the index to put in back in as a column in the df_tot dataframe
df_tot.reset_index(inplace = True)

# rename columns
df_tot.columns = ['year', 'total']

# view the final dataframe
df_tot.head()
```

Out[20]:

	year	total
0	1980	99137
1	1981	110563
2	1982	104271
3	1983	75550
4	1984	73417

Step 2: Plot the data. In `Matplotlib`, we can create a `scatter` plot set by passing in `kind='scatter'` as plot argument. We will also need to pass in `x` and `y` keywords to specify the columns that go on the x- and the y-axis.

In [21]:
```
df_tot.plot(kind='scatter', x='year', y='total', figsize=(10, 6), color='darkblue')

plt.title('Total Immigration to Canada from 1980 - 2013')
plt.xlabel('Year')
plt.ylabel('Number of Immigrants')

plt.show()
```



Notice how the scatter plot does not connect the data points together. We can clearly observe an upward trend in the data: as the years go by, the total number of immigrants increases. We can mathematically analyze this upward trend using a regression line (line of best fit).

So let's try to plot a linear line of best fit, and use it to predict the number of immigrants in 2015.

Step 1: Get the equation of line of best fit. We will use **Numpy**'s `polyfit()` method by passing in the following:

- `x` : x-coordinates of the data.
- `y` : y-coordinates of the data.
- `deg` : Degree of fitting polynomial. 1 = linear, 2 = quadratic, and so on.

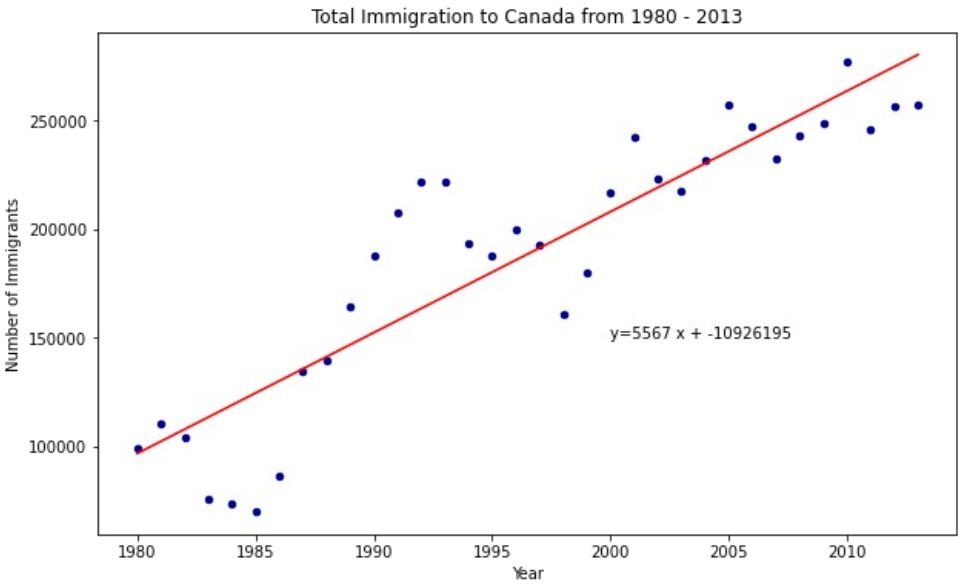
In [22]:
```
x = df_tot['year']      # year on x-axis
y = df_tot['total']     # total on y-axis
fit = np.polyfit(x, y, deg=1)

fit
```

Out[22]:
```
array([ 5.56709228e+03, -1.09261952e+07])
```

The output is an array with the polynomial coefficients, highest powers first. Since we are plotting a linear regression `y= a * x + b`, our output has 2 elements `[5.56709228e+03, -1.09261952e+07]` with the the slope in position 0 and intercept in position 1.

Step 2: Plot the regression line on the `scatter plot`.

In [24]:
```
df_tot.plot(kind='scatter', x='year', y='total', figsize=(10, 6), color='darkblue')

plt.title('Total Immigration to Canada from 1980 - 2013')
plt.xlabel('Year')
plt.ylabel('Number of Immigrants')

# plot line of best fit
plt.plot(x, fit[0] * x + fit[1], color='red') # recall that x is the Years
plt.annotate('y={0:.0f} x + {1:.0f}'.format(fit[0], fit[1]), xy=(2000, 150000))

plt.show()

# print out the line of best fit
'No. Immigrants = {0:.0f} * Year + {1:.0f}'.format(fit[0], fit[1])
```



Out[24]:
```
'No. Immigrants = 5567 * Year + -10926195'
```

Using the equation of line of best fit, we can estimate the number of immigrants in 2015:

```
No. Immigrants = 5567 * Year - 10926195
No. Immigrants = 5567 * 2015 - 10926195
No. Immigrants = 291,310
```

Question: Create a scatter plot of the total immigration from Pakistan, China, and India to Canada from 1980 to 2013?

Step 1: Get the data:

1. Create a dataframe the consists of the numbers associated with Denmark, Norway, and Sweden only. Name it **df_countries**.
2. Sum the immigration numbers across all three countries for each year and turn the result into a dataframe. Name this new dataframe **df_total**.
3. Reset the index in place.
4. Rename the columns to **year** and **total**.
5. Display the resulting dataframe.

In [26]:
```
### type your answer here
```

▶ Click here for a sample python solution

Bubble Plots

A `bubble plot` is a variation of the `scatter plot` that displays three dimensions of data (x, y, z). The data points are replaced with bubbles, and the size of the bubble is determined by the third variable `z`, also known as the weight. In `maplotlib`, we can pass in an array or scalar to the parameter `s` to `plot()`, that contains the weight of each point.

Let's start by analyzing the effect of Argentina's great depression.

Argentina suffered a great depression from 1998 to 2002, which caused widespread unemployment, riots, the fall of the government, and a default on the country's foreign debt. In terms of income, over 50% of Argentines were poor, and seven out of ten Argentine children were poor at the depth of the crisis in 2002.

Let's analyze the effect of this crisis, and compare Argentina's immigration to that of it's neighbour Brazil. Let's do that using a `bubble plot` of immigration from Brazil and Argentina for the years 1980 - 2013. We will set the weights for the bubble as the *normalized* value of the population for each year.

Step 1: Get the data for Brazil and Argentina. Like in the previous example, we will convert the `Years` to type int and include it in the dataframe.

In [32]:
```
# transposed dataframe
df_t = df[years].transpose()

# cast the Years (the index) to type int
df_t.index = map(int, df_t.index)

# let's label the index. This will automatically be the column name when we reset the index
df_t.index.name = 'Year'

# reset index to bring the Year in as a column
df_t.reset_index(inplace=True)

# view the changes
df_t.head()
```

Out[32]:

Country	Year	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Antigua and Barbuda	Argentina	Armenia	...	United States of America	Uruguay	Uzbekistan
0	1980	16	1	80	0	0	1	0	368	0	...	9378	128	0
1	1981	39	0	67	1	0	3	0	426	0	...	10030	132	0
2	1982	39	0	71	0	0	6	0	626	0	...	9074	146	0
3	1983	47	0	69	0	0	6	0	241	0	...	7100	105	0
4	1984	71	0	63	0	0	4	42	237	0	...	6661	90	0

5 rows × 196 columns

Step 2: Create the normalized weights.

There are several methods of normalizations in statistics, each with its own use. In this case, we will use feature scaling to bring all values into the range [0, 1]. The general formula is:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

where
X
is the original value,
X'
is the corresponding normalized value. The formula sets the max value in the dataset to 1, and sets the min value to 0. The rest of the data points are scaled to a value between 0-1 accordingly.

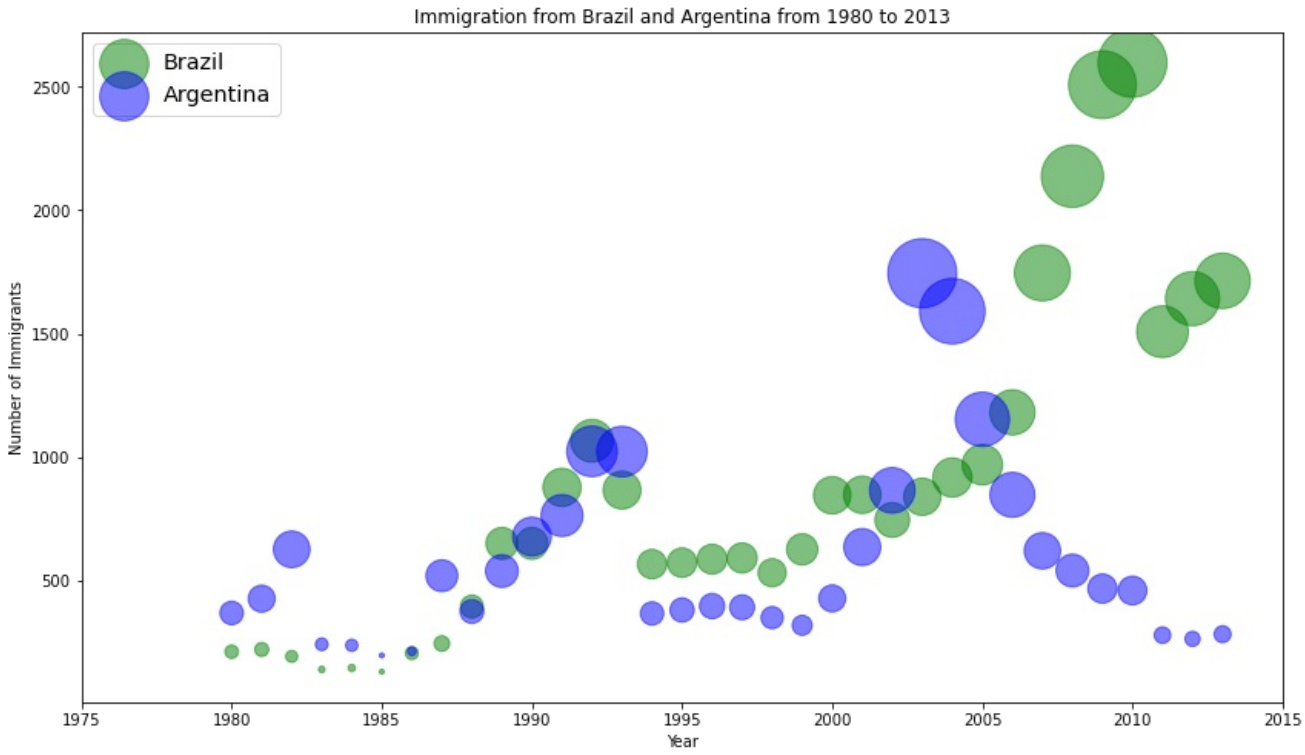
In [33]:
```
# normalize Brazil data
norm_brazil = (df_t['Brazil'] - df_t['Brazil'].min()) / (df_t['Brazil'].max() - df_t['Brazil'].min())

# normalize Argentina data
norm_argentina = (df_t['Argentina'] - df_t['Argentina'].min()) / (df_t['Argentina'].max() - df_t['Argentina'].m
```

Step 3: Plot the data.

- To plot two different scatter plots in one plot, we can include the axes one plot into the other by passing it via the `ax` parameter.
- We will also pass in the weights using the `s` parameter. Given that the normalized weights are between 0-1, they won't be visible on the plot. Therefore, we will:
  - multiply weights by 2000 to scale it up on the graph, and,
  - add 10 to compensate for the min value (which has a 0 weight and therefore scale with $\times 2000$).

In [34]:
```
# Brazil
ax0 = df_t.plot(kind='scatter',
                    x='Year',
                    y='Brazil',
                    figsize=(14, 8),
                    alpha=0.5,  # transparency
                    color='green',
                    s=norm_brazil * 2000 + 10,  # pass in weights
                    xlim=(1975, 2015)
                    )

# Argentina
ax1 = df_t.plot(kind='scatter',
                    x='Year',
                    y='Argentina',
                    alpha=0.5,
                    color="blue",
                    s=norm_argentina * 2000 + 10,
                    ax=ax0
                    )

ax0.set_ylabel('Number of Immigrants')
ax0.set_title('Immigration from Brazil and Argentina from 1980 to 2013')
ax0.legend(['Brazil', 'Argentina'], loc='upper left', fontsize='x-large')
```

Out[34]:
```
<matplotlib.legend.Legend at 0x1ee95d3fcd0>
```



The size of the bubble corresponds to the magnitude of immigrating population for that year, compared to the 1980 - 2013 data. The larger the bubble is, the more immigrants are in that year.

From the plot above, we can see a corresponding increase in immigration from Argentina during the 1998 - 2002 great depression. We can also observe a similar spike around 1985 to 1993. In fact, Argentina had suffered a great depression from 1974 to 1990, just before the onset of 1998 - 2002 great depression.

On a similar note, Brazil suffered the *Samba Effect* where the Brazilian real (currency) dropped nearly 35% in 1999. There was a fear of a South American financial crisis as many South American countries were heavily dependent on industrial exports from Brazil. The Brazilian government subsequently adopted an austerity program, and the economy slowly recovered over the years, culminating in a surge in 2010. The immigration data reflect these events.

Question: Previously in this tutorial, we created box plots to compare immigration from China and Pakistan to Canada. Create bubble plots of immigration from China and Pakistan to visualize any differences with time from 1980 to 2013. You can use **df_t** that we defined and used in the previous example.

▶ Click here for a sample python solution

Thank you

Author

Moazzam Ali

In [ ]: