DV0101EN-2-2-1-Area-Plots-Histograms-and-Bar-Charts-py-v2.0

May 28, 2020

Area Plots, Histograms, and Bar Plots

0.1 Introduction

In this lab, we will continue exploring the Matplotlib library and will learn how to create additional plots, namely area plots, histograms, and bar charts.

0.2 Table of Contents

1 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 that we are exploring 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. For this lesson, we will focus on the Canadian Immigration data.

2 Downloading and Prepping Data

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

[1]:

```
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
```

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

[2]:
```
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 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.

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

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

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

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

[5 rows x 43 columns]
```

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

[4]:
```
# 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` lab for the rational and detailed description of the changes.

1. Clean up the dataset to remove columns that are not informative to us for visualization (eg. Type, AREA, REG).

[5]:

```
df_can.drop(['AREA', 'REG', 'DEV', 'Type', 'Coverage'], axis=1, inplace=True)

# let's view the first five elements and see how the dataframe was changed
df_can.head()
```

[5]:

```
           OdName AreaName           RegName             DevName  1980  1981  \
0     Afghanistan     Asia     Southern Asia  Developing regions    16    39
1         Albania   Europe   Southern Europe   Developed regions     1     0
2         Algeria   Africa   Northern Africa  Developing regions    80    67
3  American Samoa  Oceania         Polynesia  Developing regions     0     1
4         Andorra   Europe   Southern Europe   Developed regions     0     0

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

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

[5 rows x 38 columns]
```

Notice how the columns Type, Coverage, AREA, REG, and DEV got removed from the dataframe.

2. Rename some of the columns so that they make sense.

[6]:

```
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent','RegName':
 ↪'Region'}, inplace=True)

# let's view the first five elements and see how the dataframe was changed
df_can.head()
```

[6]:

```
          Country Continent           Region             DevName  1980  1981  \
0     Afghanistan      Asia    Southern Asia  Developing regions    16    39
```

```
1         Albania    Europe  Southern Europe   Developed regions     1     0
2         Algeria    Africa  Northern Africa  Developing regions    80    67
3  American Samoa   Oceania        Polynesia  Developing regions     0     1
4         Andorra    Europe  Southern Europe   Developed regions     0     0

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

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

[5 rows x 38 columns]
```

Notice how the column names now make much more sense, even to an outsider.

3. For consistency, ensure that all column labels of type string.

[7]:
```python
# let's examine the types of the column labels
all(isinstance(column, str) for column in df_can.columns)
```

[7]:
```
False
```

Notice how the above line of code returned *False* when we tested if all the column labels are of type **string**. So let's change them all to **string** type.

[8]:
```python
df_can.columns = list(map(str, df_can.columns))

# let's check the column labels types now
all(isinstance(column, str) for column in df_can.columns)
```

[8]:
```
True
```

4. Set the country name as index - useful for quickly looking up countries using .loc method.

[9]:
```python
df_can.set_index('Country', inplace=True)

# let's view the first five elements and see how the dataframe was changed
df_can.head()
```

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

                     1982  1983  1984  1985  1986  ...  2004  2005  2006  2007  \
     Country                                       ...
     Afghanistan       39    47    71   340   496  ...  2978  3436  3009  2652
     Albania            0     0     0     0     1  ...  1450  1223   856   702
     Algeria           71    69    63    44    69  ...  3616  3626  4807  3623
     American Samoa     0     0     0     0     0  ...     0     0     1     0
     Andorra            0     0     0     0     2  ...     0     0     1     1

                     2008  2009  2010  2011  2012  2013
     Country
     Afghanistan     2111  1746  1758  2203  2635  2004
     Albania          560   716   561   539   620   603
     Algeria         4005  5393  4752  4325  3774  4331
     American Samoa     0     0     0     0     0     0
     Andorra            0     0     0     0     1     1

     [5 rows x 37 columns]
```

Notice how the country names now serve as indices.

5. Add total column.

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

      # let's view the first five elements and see how the dataframe was changed
      df_can.head()
```

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

                      1982  1983  1984  1985  1986  ...  2005  2006  2007  2008  \
      Country                                       ...
      Afghanistan       39    47    71   340   496  ...  3436  3009  2652  2111
      Albania            0     0     0     0     1  ...  1223   856   702   560
      Algeria           71    69    63    44    69  ...  3626  4807  3623  4005
```

```
American Samoa     0     0     0     0     0  …     0     1     0     0
Andorra            0     0     0     0     2  …     0     1     1     0

                2009  2010  2011  2012  2013  Total
Country
Afghanistan     1746  1758  2203  2635  2004  58639
Albania          716   561   539   620   603  15699
Algeria         5393  4752  4325  3774  4331  69439
American Samoa     0     0     0     0     0      6
Andorra            0     0     0     1     1     15

[5 rows x 38 columns]
```

Now the dataframe has an extra column that presents the total number of immigrants from each country in the dataset from 1980 - 2013. So if we print the dimension of the data, we get:

[11]:

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

```
data dimensions: (195, 38)
```

So now our dataframe has 38 columns instead of 37 columns that we had before.

[12]:

```
# finally, let's create a list of years from 1980 - 2013
# this will come in handy when we start plotting the data
years = list(map(str, range(1980, 2014)))

years
```

[12]:

```
['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']
```

3 Visualizing Data using Matplotlib

Import `Matplotlib` and **Numpy**.

[13]:

```
# use the inline backend to generate the plots within the browser
%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:  3.1.1
```

4 Area Plots

In the last module, we created a line plot that visualized the top 5 countries that contribued the most immigrants to Canada from 1980 to 2013. With a little modification to the code, we can visualize this plot as a cumulative plot, also knows as a **Stacked Line Plot** or **Area plot**.

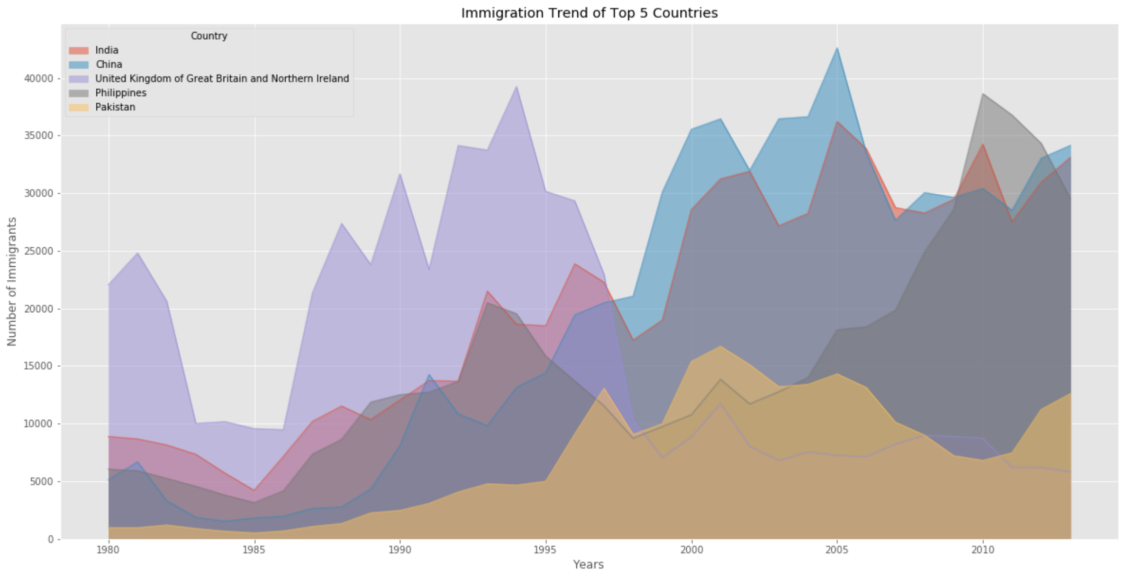
[14]:

```
df_can.sort_values(['Total'], ascending=False, axis=0, inplace=True)

# get the top 5 entries
df_top5 = df_can.head()

# transpose the dataframe
df_top5 = df_top5[years].transpose()

df_top5.head()
```

```
[14]: Country  India  China  United Kingdom of Great Britain and Northern Ireland  \
      1980      8880   5123                                               22045
      1981      8670   6682                                               24796
      1982      8147   3308                                               20620
      1983      7338   1863                                               10015
      1984      5704   1527                                               10170

      Country  Philippines  Pakistan
      1980            6051       978
      1981            5921       972
      1982            5249      1201
      1983            4562       900
      1984            3801       668
```

Area plots are stacked by default. And to produce a stacked area plot, each column must be either all positive or all negative values (any NaN values will defaulted to 0). To produce an unstacked plot, pass `stacked=False`.

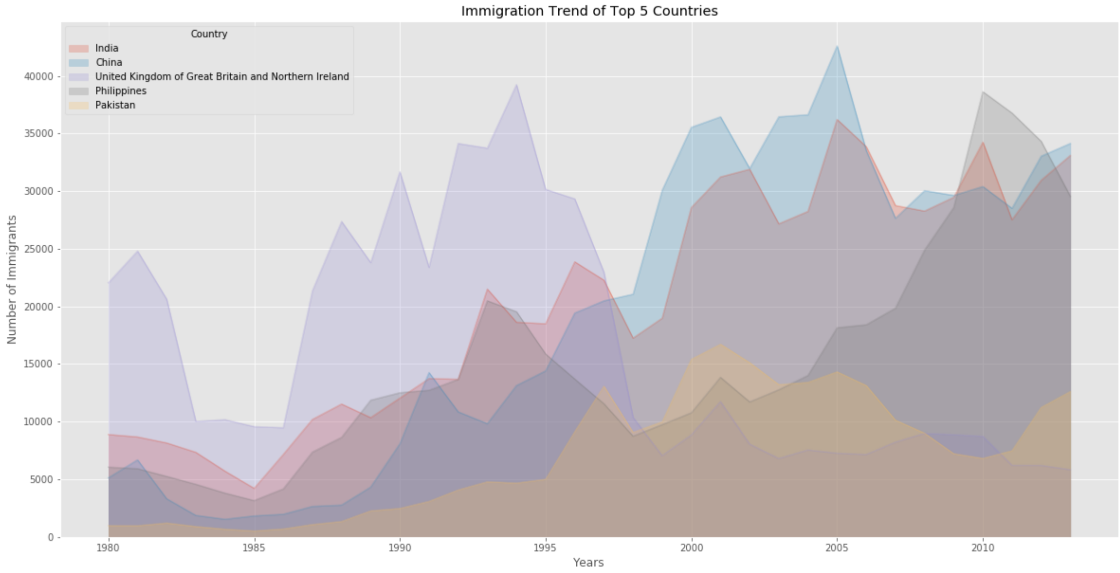
```
[15]: df_top5.index = df_top5.index.map(int) # let's change the index values of␣
       ↪df_top5 to type integer for plotting
      df_top5.plot(kind='area',
                   stacked=False,
                   figsize=(20, 10), # pass a tuple (x, y) size
                   )

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

      plt.show()
```



The unstacked plot has a default transparency (alpha value) at 0.5. We can modify this value by passing in the alpha parameter.

[16]:

```
df_top5.plot(kind='area',
             alpha=0.25, # 0-1, default value a= 0.5
             stacked=False,
             figsize=(20, 10),
            )

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

plt.show()
```



4.0.1 Two types of plotting

As we discussed in the video lectures, there are two styles/options of ploting with `matplotlib`. Plotting using the Artist layer and plotting using the scripting layer.

Option 1: Scripting layer (procedural method) - using matplotlib.pyplot as 'plt'

You can use `plt` i.e. `matplotlib.pyplot` and add more elements by calling different methods procedurally; for example, `plt.title(...)` to add title or `plt.xlabel(...)` to add label to the x-axis.

```
# Option 1: This is what we have been using so far
df_top5.plot(kind='area', alpha=0.35, figsize=(20, 10))
```

```
plt.title('Immigration trend of top 5 countries')
plt.ylabel('Number of immigrants')
plt.xlabel('Years')
```

Option 2: Artist layer (Object oriented method) - using an Axes instance from Matplotlib (preferred)

You can use an `Axes` instance of your current plot and store it in a variable (eg. `ax`). You can add more elements by calling methods with a little change in syntax (by adding "*set_*" to the previous methods). For example, use `ax.set_title()` instead of `plt.title()` to add title, or `ax.set_xlabel()` instead of `plt.xlabel()` to add label to the x-axis.

This option sometimes is more transparent and flexible to use for advanced plots (in particular when having multiple plots, as you will see later).

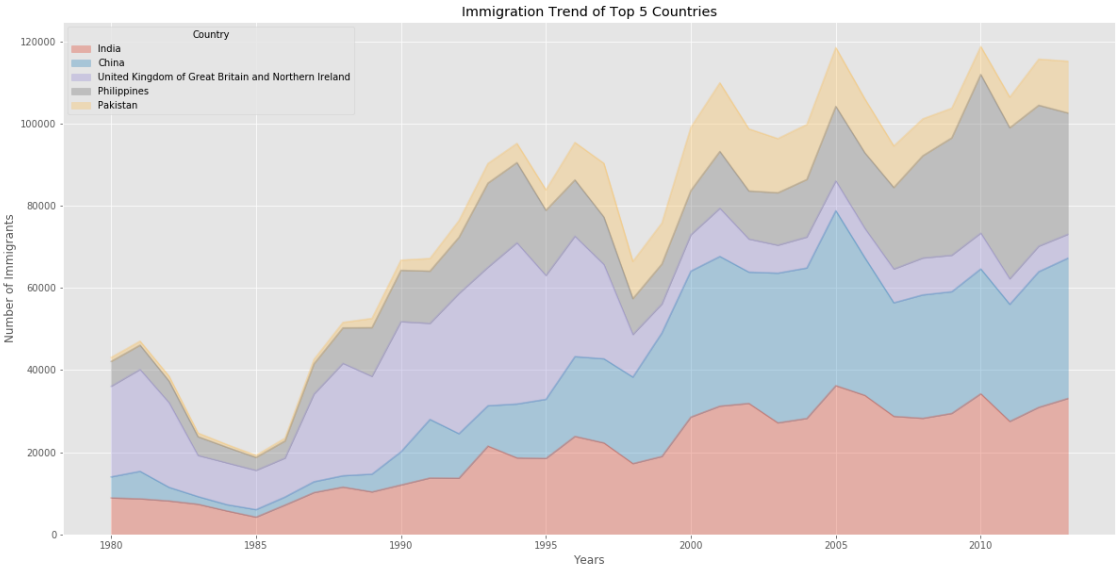
In this course, we will stick to the **scripting layer**, except for some advanced visualizations where we will need to use the **artist layer** to manipulate advanced aspects of the plots.

[17]:

```
# option 2: preferred option with more flexibility
ax = df_top5.plot(kind='area', alpha=0.35, figsize=(20, 10))

ax.set_title('Immigration Trend of Top 5 Countries')
ax.set_ylabel('Number of Immigrants')
ax.set_xlabel('Years')
```

[17]:

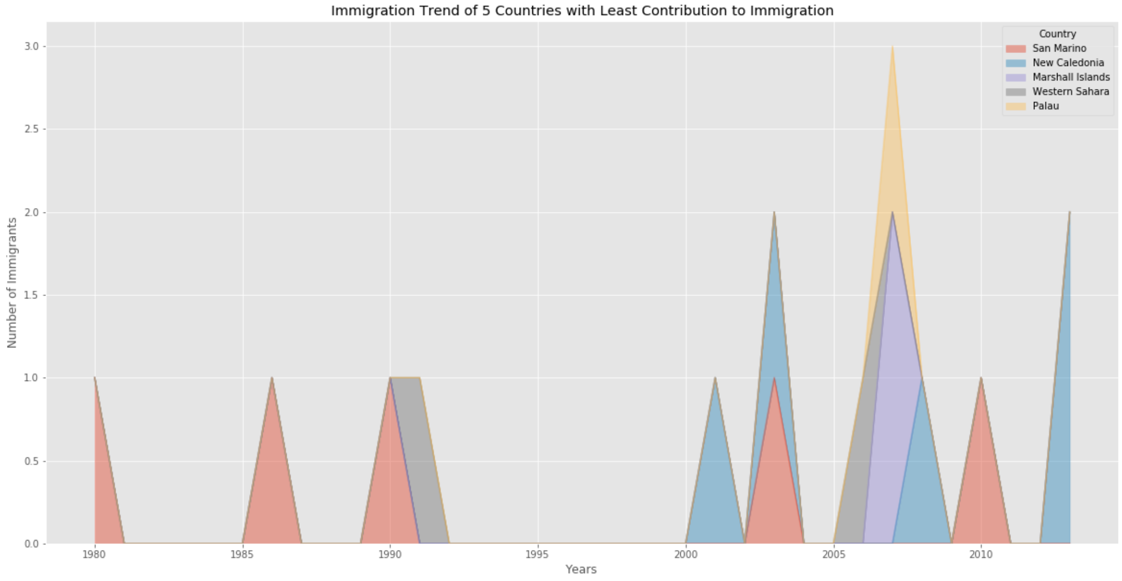
```
Text(0.5, 0, 'Years')
```



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

[23]:

```
### type your answer here
df_last5 = df_can.tail()
df_last5 = df_last5[years].transpose()

df_last5.index = df_last5.index.map(int)
df_last5.plot(kind='area', alpha=0.45, figsize=(20,10))

plt.title('Immigration Trend of 5 Countries with Least Contribution to␣
 ↪Immigration')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')

plt.show()
```



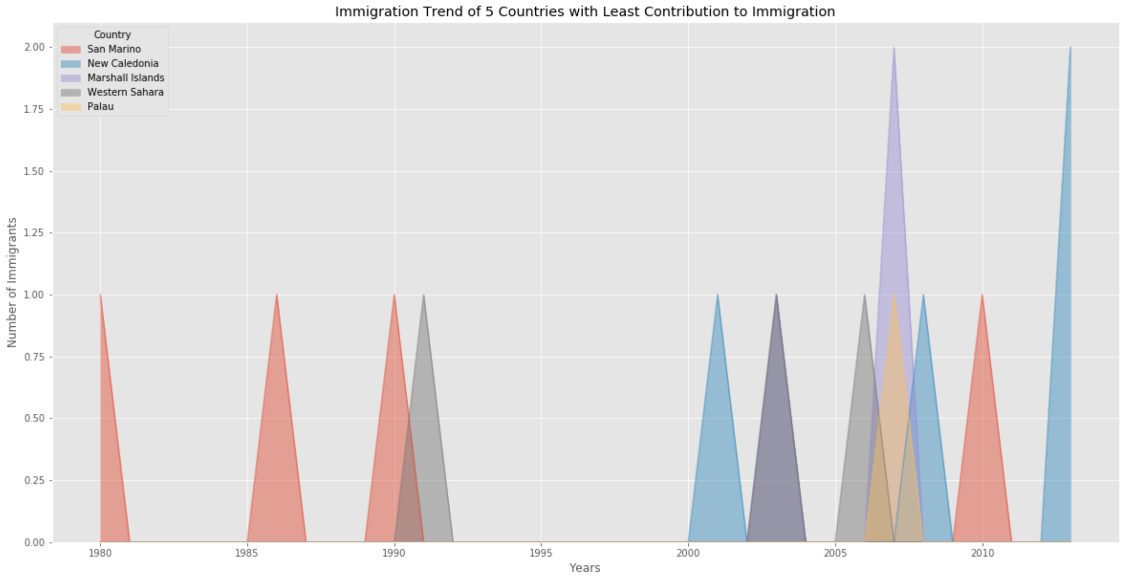
Double-click **here** for the solution.

Question: Use the artist layer to create an unstacked area plot of the 5 countries that contributed the least to immigration to Canada **from** 1980 to 2013. Use a transparency value of 0.55.

[25]:

```
### type your answer here

ax = df_last5.plot(kind='area', stacked=False, alpha=0.45, figsize=(20,10))

ax.set_title('Immigration Trend of 5 Countries with Least Contribution to␣
 ↪Immigration')
ax.set_ylabel('Number of Immigrants')
ax.set_xlabel('Years')
```

```
[25]: Text(0.5, 0, 'Years')
```



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

5 Histograms

A histogram is a way of representing the *frequency* distribution of numeric dataset. The way it works is it partitions the x-axis into *bins*, assigns each data point in our dataset to a bin, and then counts the number of data points that have been assigned to each bin. So the y-axis is the frequency or the number of data points in each bin. Note that we can change the bin size and usually one needs to tweak it so that the distribution is displayed nicely.

Question: What is the frequency distribution of the number (population) of new immigrants from the various countries to Canada in 2013?

Before we proceed with creating the histogram plot, let's first examine the data split into intervals. To do this, we will us **Numpy**'s `histrogram` method to get the bin ranges and frequency counts as follows:

```
[26]: # let's quickly view the 2013 data
      df_can['2013'].head()
```

```
[26]: Country
      India                                                   33087
      China                                                   34129
      United Kingdom of Great Britain and Northern Ireland     5827
      Philippines                                             29544
      Pakistan                                                12603
      Name: 2013, dtype: int64
```

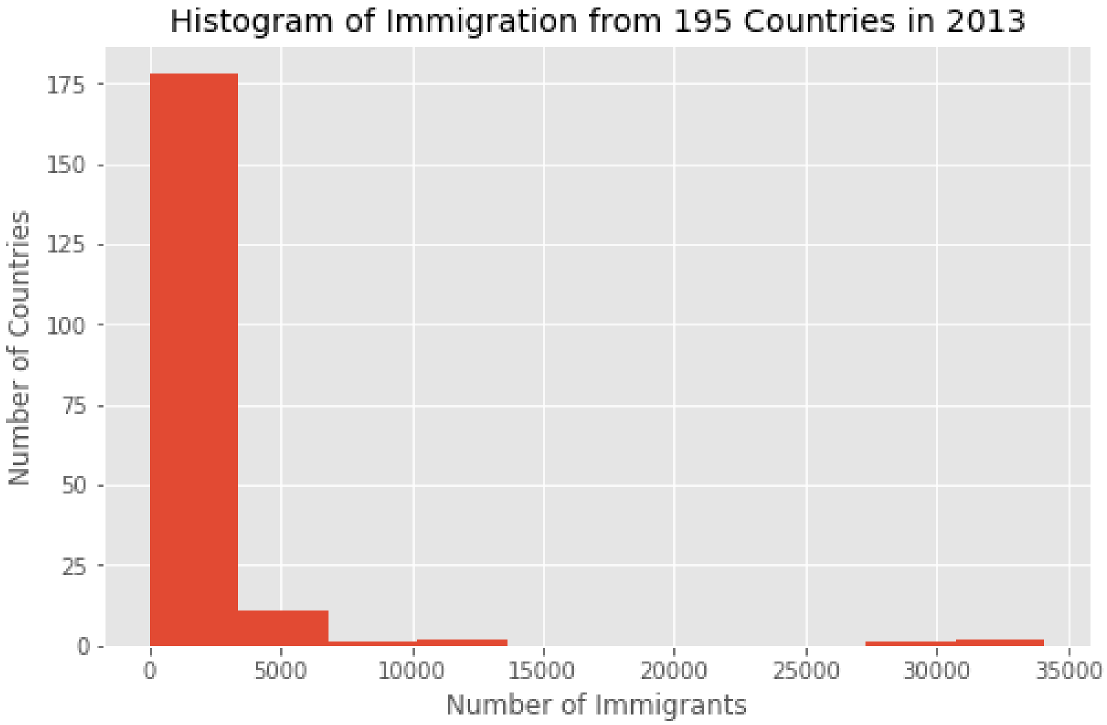
[27]:
```
# np.histogram returns 2 values
count, bin_edges = np.histogram(df_can['2013'])

print(count) # frequency count
print(bin_edges) # bin ranges, default = 10 bins
```

```
[178  11   1   2   0   0   0   0   1   2]
[    0.    3412.9  6825.8 10238.7 13651.6 17064.5 20477.4 23890.3 27303.2
 30716.1 34129. ]
```

By default, the `histrogram` method breaks up the dataset into 10 bins. The figure below summarizes the bin ranges and the frequency distribution of immigration in 2013. We can see that in 2013: * 178 countries contributed between 0 to 3412.9 immigrants * 11 countries contributed between 3412.9 to 6825.8 immigrants * 1 country contributed between 6285.8 to 10238.7 immigrants, and so on..

We can easily graph this distribution by passing `kind=hist` to `plot()`.

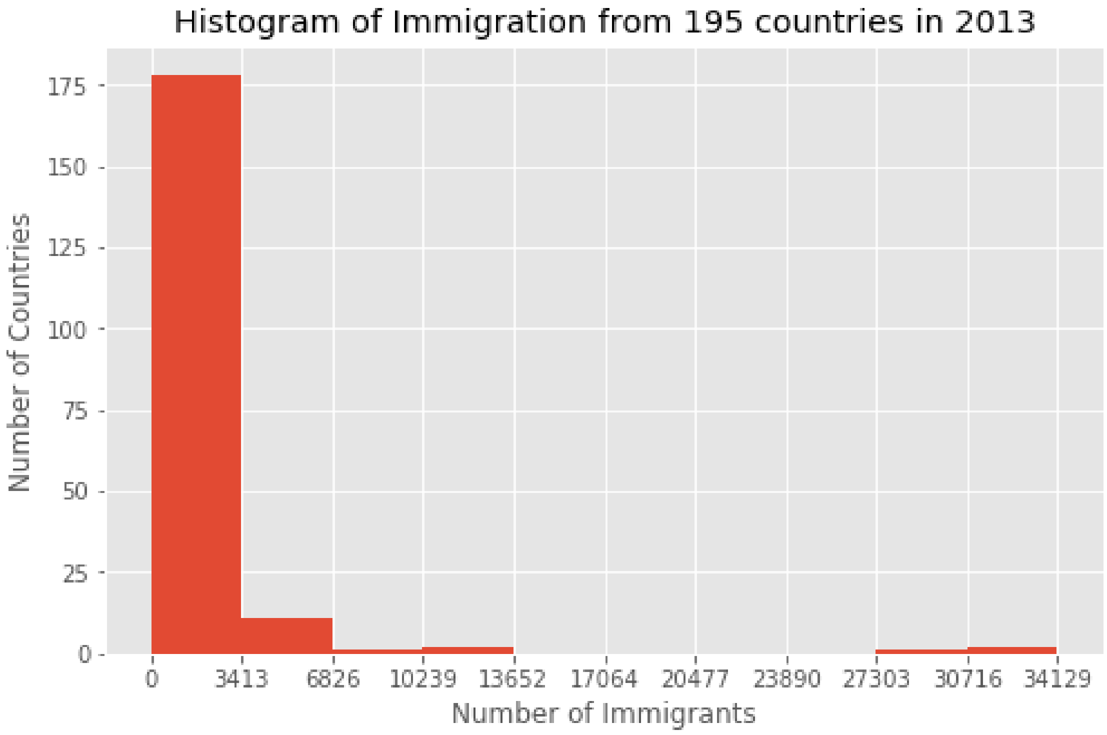
[28]:
```
df_can['2013'].plot(kind='hist', figsize=(8, 5))

plt.title('Histogram of Immigration from 195 Countries in 2013') # add a title␣
↪to the histogram
plt.ylabel('Number of Countries') # add y-label
plt.xlabel('Number of Immigrants') # add x-label

plt.show()
```



In the above plot, the x-axis represents the population range of immigrants in intervals of 3412.9. The y-axis represents the number of countries that contributed to the aforementioned population.

Notice that the x-axis labels do not match with the bin size. This can be fixed by passing in a `xticks` keyword that contains the list of the bin sizes, as follows:

[29]:
```python
# 'bin_edges' is a list of bin intervals
count, bin_edges = np.histogram(df_can['2013'])

df_can['2013'].plot(kind='hist', figsize=(8, 5), xticks=bin_edges)

plt.title('Histogram of Immigration from 195 countries in 2013') # add a title to the histogram
plt.ylabel('Number of Countries') # add y-label
plt.xlabel('Number of Immigrants') # add x-label

plt.show()
```



Side Note: We could use `df_can['2013'].plot.hist()`, instead. In fact, throughout this lesson, using `some_data.plot(kind='type_plot', ...)` is equivalent to `some_data.plot.type_plot(...)`. That is, passing the type of the plot as argument or method behaves the same.

See the *pandas* documentation for more info http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.plot.html.

We can also plot multiple histograms on the same plot. For example, let's try to answer the following questions using a histogram.

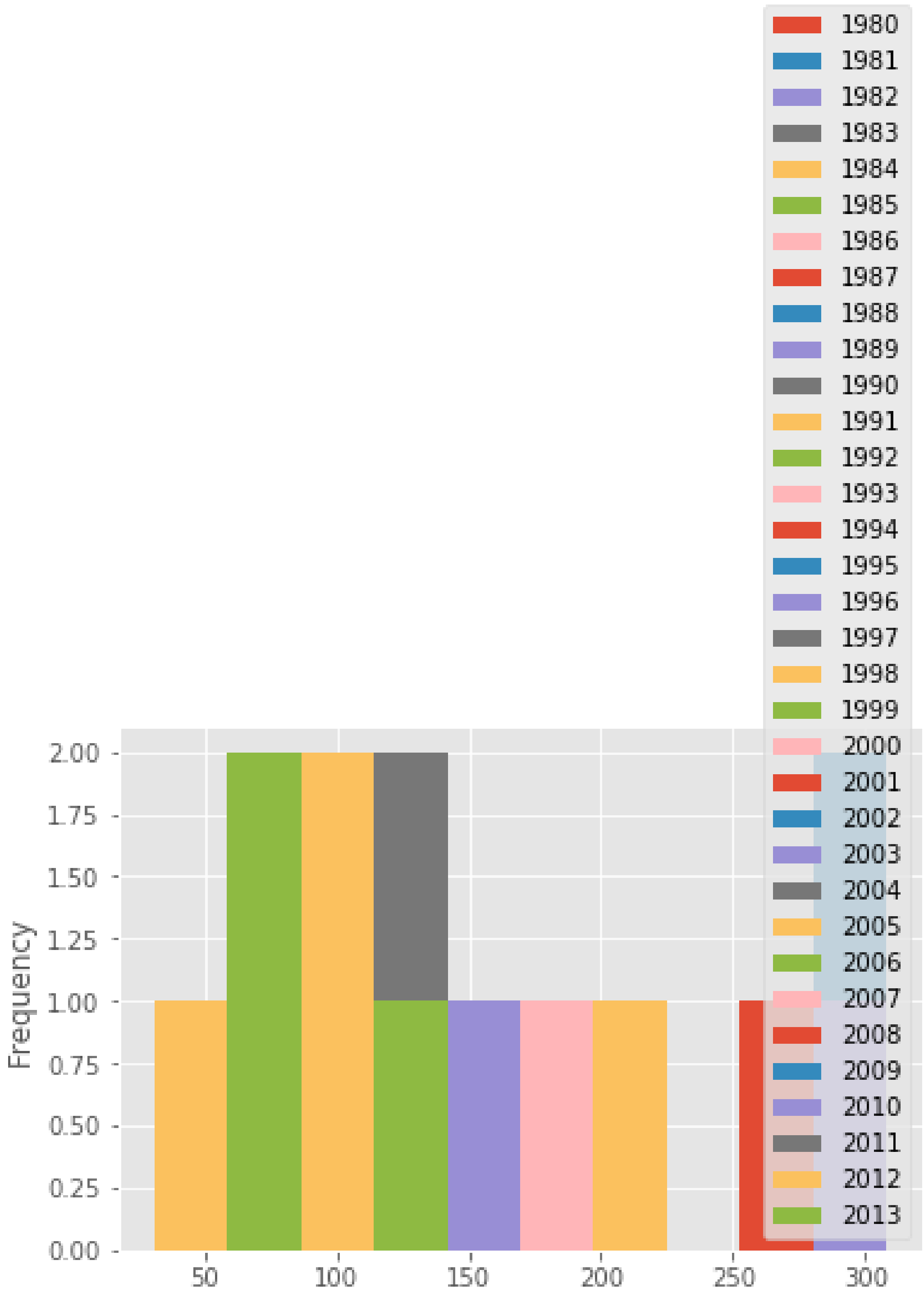
Question: What is the immigration distribution for Denmark, Norway, and Sweden for years 1980 - 2013?

```
[30]: # let's quickly view the dataset
      df_can.loc[['Denmark', 'Norway', 'Sweden'], years]
```

```
[30]:          1980  1981  1982  1983  1984  1985  1986  1987  1988  1989  …  \
      Country                                                             …
      Denmark   272   293   299   106    93    73    93   109   129   129  …
      Norway    116    77   106    51    31    54    56    80    73    76  …
      Sweden    281   308   222   176   128   158   187   198   171   182  …

               2004  2005  2006  2007  2008  2009  2010  2011  2012  2013
      Country
      Denmark    89    62   101    97   108    81    92    93    94    81
      Norway     73    57    53    73    66    75    46    49    53    59
      Sweden    129   205   139   193   165   167   159   134   140   140

      [3 rows x 34 columns]
```

```
[31]: # generate histogram
      df_can.loc[['Denmark', 'Norway', 'Sweden'], years].plot.hist()
```

```
[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f52ed0e3d30>
```



That does not look right!

Don’t worry, you’ll often come across situations like this when creating plots. The solution often lies in how the underlying dataset is structured.

Instead of plotting the population frequency distribution of the population for the 3 countries, *pandas* instead plotted the population frequency distribution for the `years`.

This can be easily fixed by first transposing the dataset, and then plotting as shown below.

[32]:

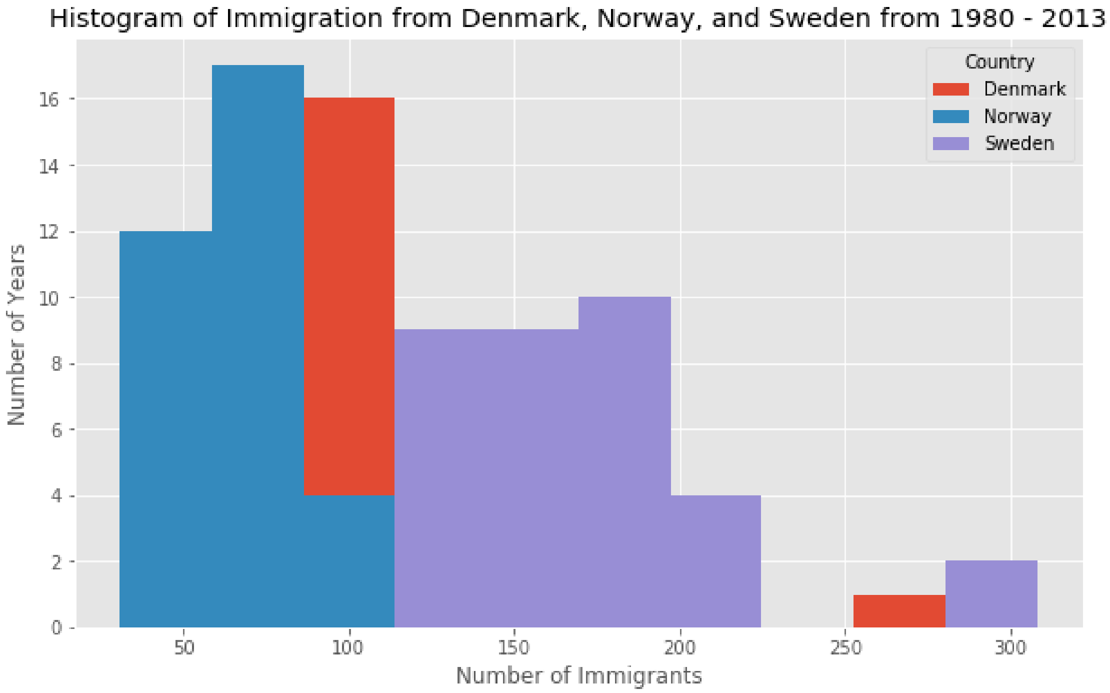
```
# transpose dataframe
df_t = df_can.loc[['Denmark', 'Norway', 'Sweden'], years].transpose()
df_t.head()
```

[32]:

```
Country  Denmark  Norway  Sweden
1980         272     116     281
1981         293      77     308
1982         299     106     222
1983         106      51     176
1984          93      31     128
```

[33]:

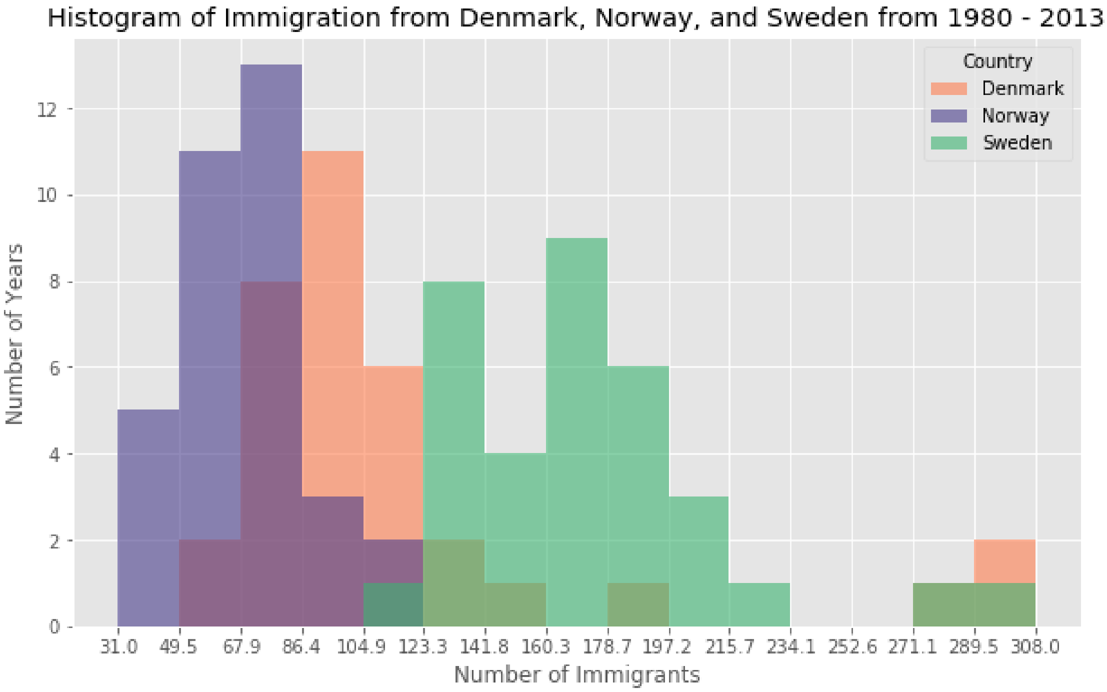
```
# generate histogram
df_t.plot(kind='hist', figsize=(10, 6))

plt.title('Histogram of Immigration from Denmark, Norway, and Sweden from 1980␣
 ↪- 2013')
plt.ylabel('Number of Years')
plt.xlabel('Number of Immigrants')

plt.show()
```



Let's make a few modifications to improve the impact and aesthetics of the previous plot: * increase the bin size to 15 by passing in `bins` parameter * set transparency to 60% by passing in `alpha` paramemter * label the x-axis by passing in `x-label` paramater * change the colors of the plots by passing in `color` parameter

[34]:

```
# let's get the x-tick values
count, bin_edges = np.histogram(df_t, 15)

# un-stacked histogram
df_t.plot(kind ='hist',
          figsize=(10, 6),
          bins=15,
          alpha=0.6,
          xticks=bin_edges,
          color=['coral', 'darkslateblue', 'mediumseagreen']
         )

plt.title('Histogram of Immigration from Denmark, Norway, and Sweden from 1980␣
 ↪- 2013')
plt.ylabel('Number of Years')
plt.xlabel('Number of Immigrants')

plt.show()
```



Tip: For a full listing of colors available in Matplotlib, run the following code in your python shell:

```
import matplotlib
for name, hex in matplotlib.colors.cnames.items():
    print(name, hex)
```

If we do no want the plots to overlap each other, we can stack them using the `stacked` paramemter. Let's also adjust the min and max x-axis labels to remove the extra gap on the edges of the plot. We can pass a tuple (min,max) using the `xlim` paramater, as show below.

[35]:
```
import matplotlib
for name, hex in matplotlib.colors.cnames.items():
    print(name, hex)
```

```
aliceblue #F0F8FF
antiquewhite #FAEBD7
aqua #00FFFF
aquamarine #7FFFD4
azure #F0FFFF
beige #F5F5DC
bisque #FFE4C4
black #000000
blanchedalmond #FFEBCD
blue #0000FF
blueviolet #8A2BE2
brown #A52A2A
burlywood #DEB887
cadetblue #5F9EA0
chartreuse #7FFF00
chocolate #D2691E
coral #FF7F50
cornflowerblue #6495ED
cornsilk #FFF8DC
crimson #DC143C
cyan #00FFFF
darkblue #00008B
darkcyan #008B8B
darkgoldenrod #B8860B
darkgray #A9A9A9
darkgreen #006400
darkgrey #A9A9A9
darkkhaki #BDB76B
darkmagenta #8B008B
darkolivegreen #556B2F
darkorange #FF8C00
darkorchid #9932CC
darkred #8B0000
darksalmon #E9967A
darkseagreen #8FBC8F
darkslateblue #483D8B
```

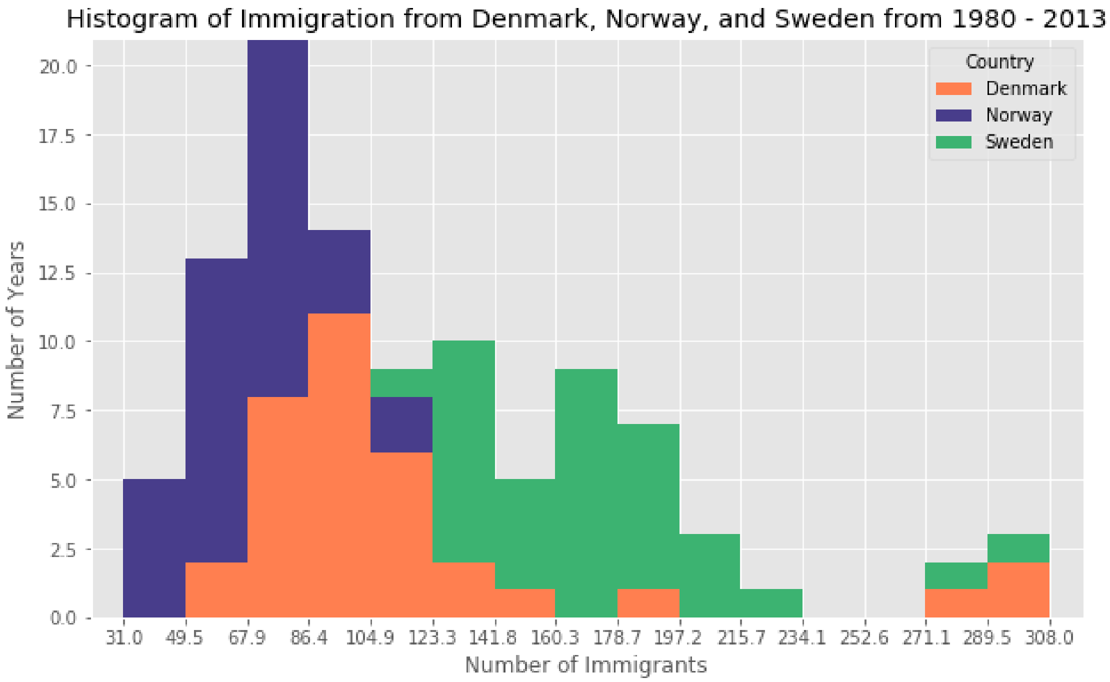
```
darkslategray #2F4F4F
darkslategrey #2F4F4F
darkturquoise #00CED1
darkviolet #9400D3
deeppink #FF1493
deepskyblue #00BFFF
dimgray #696969
dimgrey #696969
dodgerblue #1E90FF
firebrick #B22222
floralwhite #FFFAF0
forestgreen #228B22
fuchsia #FF00FF
gainsboro #DCDCDC
ghostwhite #F8F8FF
gold #FFD700
goldenrod #DAA520
gray #808080
green #008000
greenyellow #ADFF2F
grey #808080
honeydew #F0FFF0
hotpink #FF69B4
indianred #CD5C5C
indigo #4B0082
ivory #FFFFF0
khaki #F0E68C
lavender #E6E6FA
lavenderblush #FFF0F5
lawngreen #7CFC00
lemonchiffon #FFFACD
lightblue #ADD8E6
lightcoral #F08080
lightcyan #E0FFFF
lightgoldenrodyellow #FAFAD2
lightgray #D3D3D3
lightgreen #90EE90
lightgrey #D3D3D3
lightpink #FFB6C1
lightsalmon #FFA07A
lightseagreen #20B2AA
lightskyblue #87CEFA
lightslategray #778899
lightslategrey #778899
lightsteelblue #B0C4DE
lightyellow #FFFFE0
lime #00FF00
limegreen #32CD32
```

```
linen #FAF0E6
magenta #FF00FF
maroon #800000
mediumaquamarine #66CDAA
mediumblue #0000CD
mediumorchid #BA55D3
mediumpurple #9370DB
mediumseagreen #3CB371
mediumslateblue #7B68EE
mediumspringgreen #00FA9A
mediumturquoise #48D1CC
mediumvioletred #C71585
midnightblue #191970
mintcream #F5FFFA
mistyrose #FFE4E1
moccasin #FFE4B5
navajowhite #FFDEAD
navy #000080
oldlace #FDF5E6
olive #808000
olivedrab #6B8E23
orange #FFA500
orangered #FF4500
orchid #DA70D6
palegoldenrod #EEE8AA
palegreen #98FB98
paleturquoise #AFEEEE
palevioletred #DB7093
papayawhip #FFEFD5
peachpuff #FFDAB9
peru #CD853F
pink #FFC0CB
plum #DDA0DD
powderblue #B0E0E6
purple #800080
rebeccapurple #663399
red #FF0000
rosybrown #BC8F8F
royalblue #4169E1
saddlebrown #8B4513
salmon #FA8072
sandybrown #F4A460
seagreen #2E8B57
seashell #FFF5EE
sienna #A0522D
silver #C0C0C0
skyblue #87CEEB
slateblue #6A5ACD
```

```
slategray #708090
slategrey #708090
snow #FFFAFA
springgreen #00FF7F
steelblue #4682B4
tan #D2B48C
teal #008080
thistle #D8BFD8
tomato #FF6347
turquoise #40E0D0
violet #EE82EE
wheat #F5DEB3
white #FFFFFF
whitesmoke #F5F5F5
yellow #FFFF00
yellowgreen #9ACD32
```

[36]:

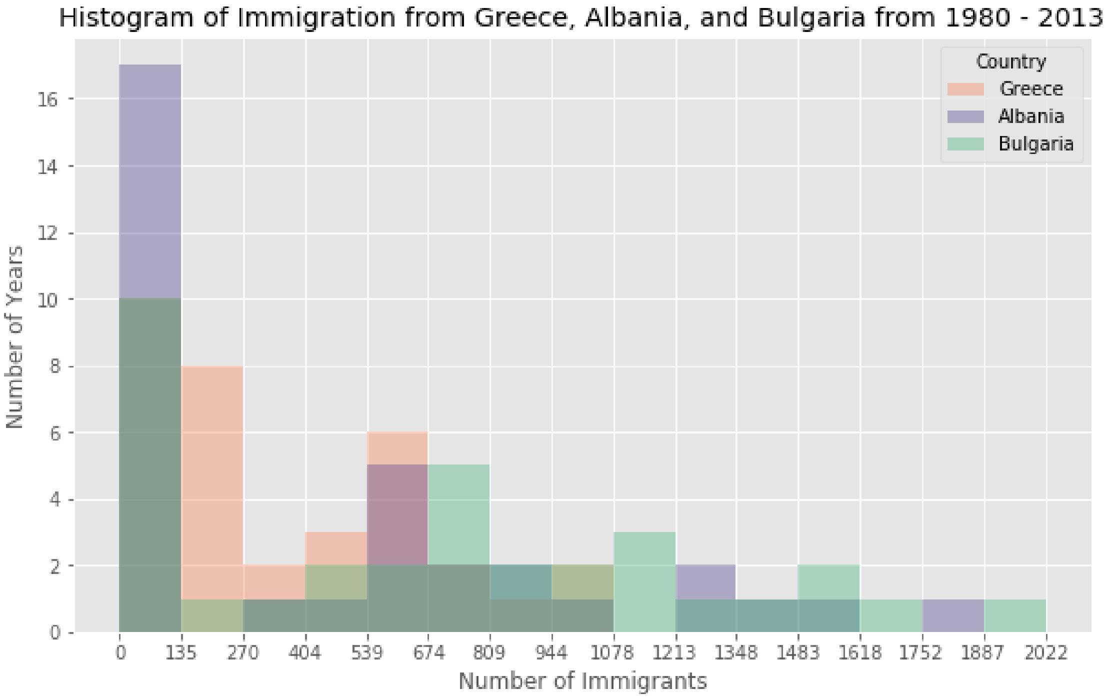
```python
count, bin_edges = np.histogram(df_t, 15)
xmin = bin_edges[0] - 10   #  first bin value is 31.0, adding buffer of 10 for aesthetic purposes
xmax = bin_edges[-1] + 10  #  last bin value is 308.0, adding buffer of 10 for aesthetic purposes

# stacked Histogram
df_t.plot(kind='hist',
          figsize=(10, 6),
          bins=15,
          xticks=bin_edges,
          color=['coral', 'darkslateblue', 'mediumseagreen'],
          stacked=True,
          xlim=(xmin, xmax)
         )

plt.title('Histogram of Immigration from Denmark, Norway, and Sweden from 1980 - 2013')
plt.ylabel('Number of Years')
plt.xlabel('Number of Immigrants')

plt.show()
```



Question: Use the scripting layer to display the immigration distribution for Greece, Albania, and Bulgaria for years 1980 - 2013? Use an overlapping plot with 15 bins and a transparency value of 0.35.

[38]:

```python
### type your answer here

df_g = df_can.loc[['Greece','Albania','Bulgaria'], years].transpose()

count, bin_edges = np.histogram(df_g, 15)

df_g.plot(kind='hist',
         figsize=(10,6),
         bins=15,
         alpha=0.36,
         xticks=bin_edges,
         color=['coral','darkslateblue','mediumseagreen'])

plt.title('Histogram of Immigration from Greece, Albania, and Bulgaria from 1980 - 2013')
plt.ylabel('Number of Years')
plt.xlabel('Number of Immigrants')

plt.show()
```



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

6 Bar Charts (Dataframe)

A bar plot is a way of representing data where the *length* of the bars represents the magnitude/size of the feature/variable. Bar graphs usually represent numerical and categorical variables grouped in intervals.

To create a bar plot, we can pass one of two arguments via `kind` parameter in `plot()`:

- `kind=bar` creates a *vertical* bar plot
- `kind=barh` creates a *horizontal* bar plot

Vertical bar plot

In vertical bar graphs, the x-axis is used for labelling, and the length of bars on the y-axis corresponds to the magnitude of the variable being measured. Vertical bar graphs are particuarly useful in analyzing time series data. One disadvantage is that they lack space for text labelling at the foot of each bar.

Let's start off by analyzing the effect of Iceland's Financial Crisis:

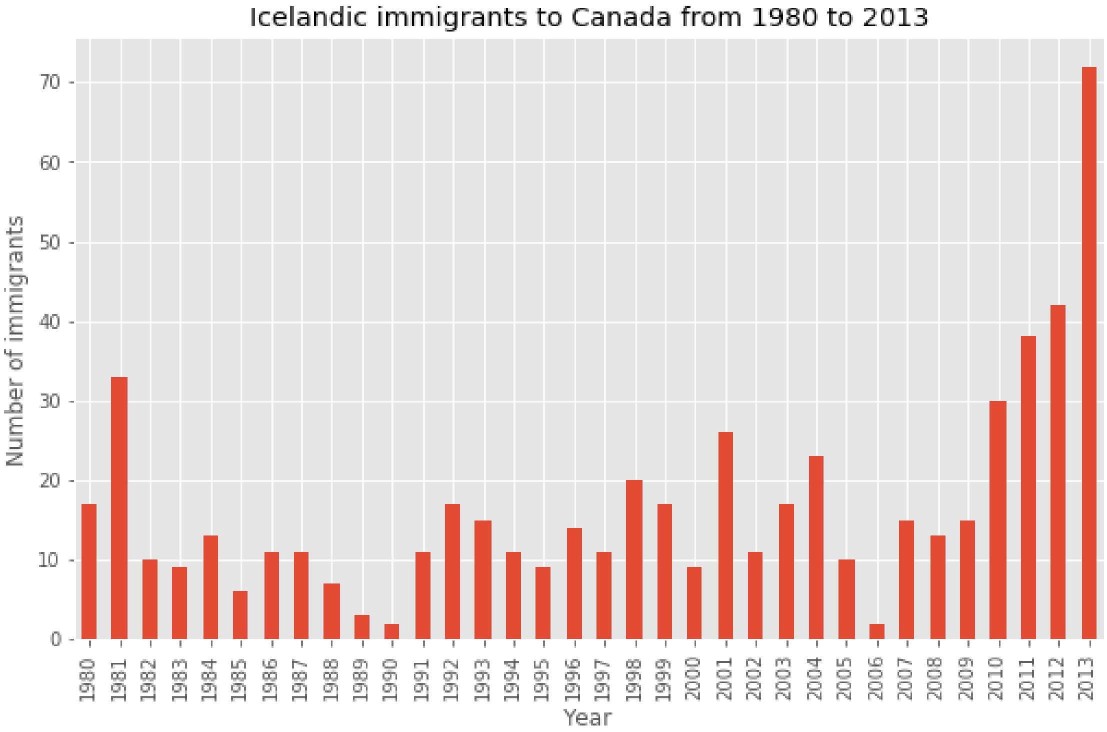
The 2008 - 2011 Icelandic Financial Crisis was a major economic and political event in Iceland. Relative to the size of its economy, Iceland's systemic banking collapse was the largest experienced by any country in economic history. The crisis led to a severe economic depression in 2008 - 2011 and significant political unrest.

Question: Let's compare the number of Icelandic immigrants (country = 'Iceland') to Canada from year 1980 to 2013.

[39]:
```
# step 1: get the data
df_iceland = df_can.loc['Iceland', years]
df_iceland.head()
```

[39]:
```
1980    17
1981    33
1982    10
1983     9
1984    13
Name: Iceland, dtype: object
```

[40]:
```
# step 2: plot data
df_iceland.plot(kind='bar', figsize=(10, 6))

plt.xlabel('Year') # add to x-label to the plot
plt.ylabel('Number of immigrants') # add y-label to the plot
plt.title('Icelandic immigrants to Canada from 1980 to 2013') # add title to␣
 ↪the plot

plt.show()
```

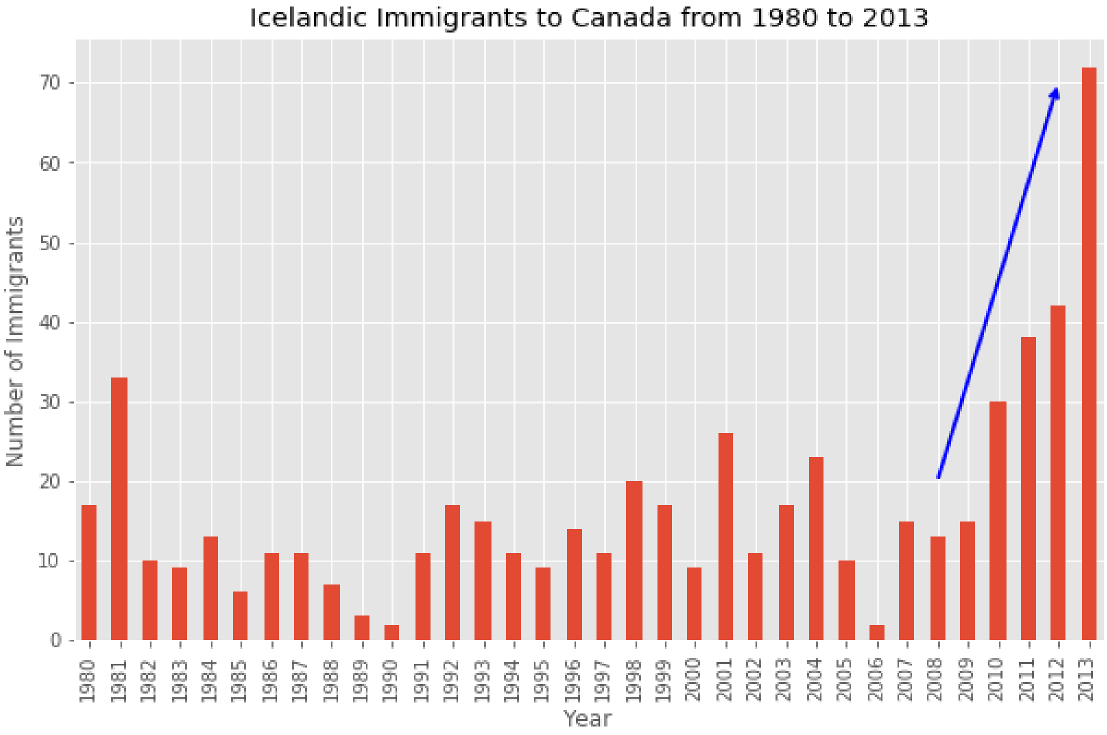


The bar plot above shows the total number of immigrants broken down by each year. We can clearly see the impact of the financial crisis; the number of immigrants to Canada started increasing rapidly after 2008.

Let's annotate this on the plot using the `annotate` method of the **scripting layer** or the **pyplot interface**. We will pass in the following parameters: - `s`: str, the text of annotation. - `xy`: Tuple specifying the (x,y) point to annotate (in this case, end point of arrow). - `xytext`: Tuple specifying the (x,y) point to place the text (in this case, start point of arrow). - `xycoords`: The coordinate system that xy is given in - 'data' uses the coordinate system of the object being annotated (default). - `arrowprops`: Takes a dictionary of properties to draw the arrow: - `arrowstyle`: Specifies the arrow style, `'->'` is standard arrow. - `connectionstyle`: Specifies the connection type. `arc3` is a straight line. - `color`: Specifes color of arror. - `lw`: Specifies the line width.

I encourage you to read the Matplotlib documentation for more details on annotations: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.annotate.

[41]:
```
df_iceland.plot(kind='bar', figsize=(10, 6), rot=90) # rotate the bars by 90␣
 ↪degrees

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

# Annotate arrow
plt.annotate('',                      # s: str. Will leave it blank for no text
             xy=(32, 70),             # place head of the arrow at point (year␣
 ↪2012 , pop 70)
             xytext=(28, 20),         # place base of the arrow at point (year␣
 ↪2008 , pop 20)
             xycoords='data',         # will use the coordinate system of the␣
 ↪object being annotated
             arrowprops=dict(arrowstyle='->', connectionstyle='arc3',␣
 ↪color='blue', lw=2)
            )

plt.show()
```

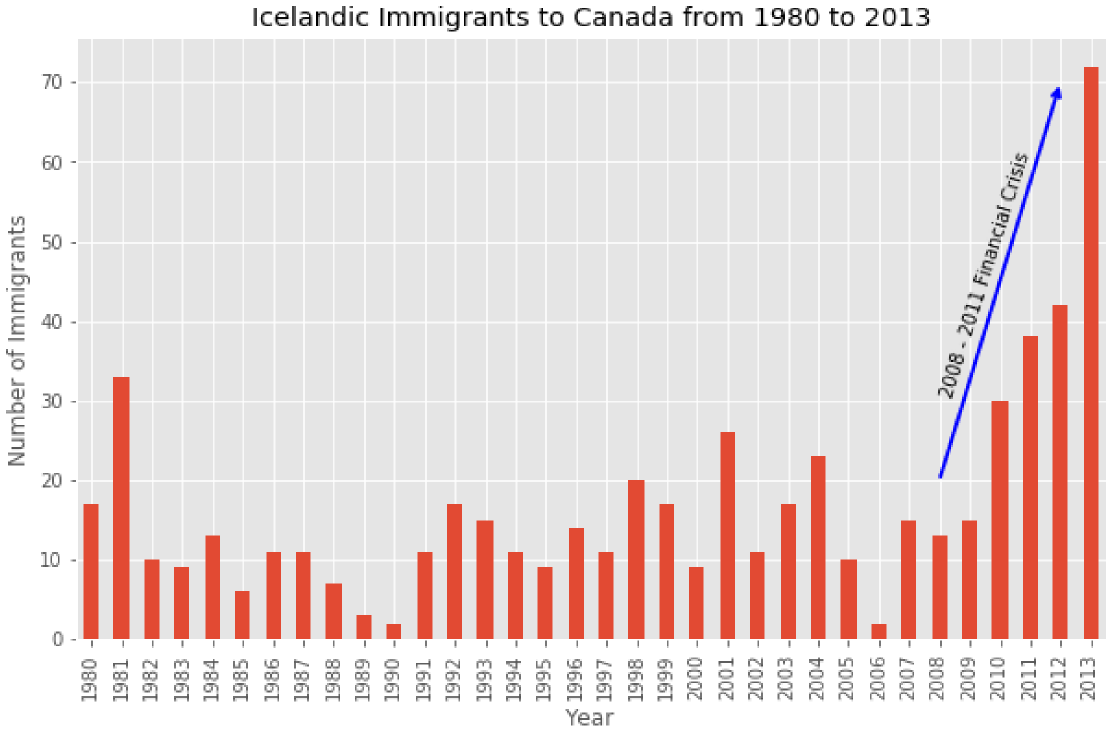


Let's also annotate a text to go over the arrow. We will pass in the following additional parameters: - `rotation`: rotation angle of text in degrees (counter clockwise) - `va`: vertical alignment of text ['center' | 'top' | 'bottom' | 'baseline'] - `ha`: horizontal alignment of text ['center' | 'right' | 'left']

[42]:

```
df_iceland.plot(kind='bar', figsize=(10, 6), rot=90)

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

# Annotate arrow
plt.annotate('',                          # s: str. will leave it blank for no text
             xy=(32, 70),                 # place head of the arrow at point (year␣
 ↪2012 , pop 70)
             xytext=(28, 20),             # place base of the arrow at point (year␣
 ↪2008 , pop 20)
             xycoords='data',             # will use the coordinate system of the␣
 ↪object being annotated
             arrowprops=dict(arrowstyle='->', connectionstyle='arc3',␣
 ↪color='blue', lw=2)
            )

# Annotate Text
```

```
plt.annotate('2008 - 2011 Financial Crisis', # text to display
             xy=(28, 30),                    # start the text at at point (year␣
 ↪2008 , pop 30)
             rotation=72.5,                  # based on trial and error to␣
 ↪match the arrow
             va='bottom',                    # want the text to be vertically␣
 ↪'bottom' aligned
             ha='left',                      # want the text to be horizontally␣
 ↪'left' algned.
            )

plt.show()
```



Horizontal Bar Plot

Sometimes it is more practical to represent the data horizontally, especially if you need more room for labelling the bars. In horizontal bar graphs, the y-axis is used for labelling, and the length of bars on the x-axis corresponds to the magnitude of the variable being measured. As you will see, there is more room on the y-axis to label categetorical variables.

Question: Using the scripting layter and the `df_can` dataset, create a *horizontal* bar plot showing the *total* number of immigrants to Canada from the top 15 countries, for the period 1980 - 2013. Label each country with the total immigrant count.

Step 1: Get the data pertaining to the top 15 countries.

[46]:

```python
### type your answer here
df_can.sort_values(by='Total', ascending=False, axis=0, inplace=True)

df_top15 = df_can.head(15)

df_top15
```

[46]:

```
Continent  \
Country
India
Asia
China
Asia
United Kingdom of Great Britain and Northern Ir…
Europe
Philippines
Asia
Pakistan
Asia
United States of America                                           Northern
America
Iran (Islamic Republic of)
Asia
Sri Lanka
Asia
Republic of Korea
Asia
Poland
Europe
Lebanon
Asia
France
Europe
Jamaica                                             Latin America and the
Caribbean
Viet Nam
Asia
Romania
Europe

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

```
United States of America                              Northern America
Iran (Islamic Republic of)                               Southern Asia
Sri Lanka                                                Southern Asia
Republic of Korea                                         Eastern Asia
Poland                                                  Eastern Europe
Lebanon                                                   Western Asia
France                                                  Western Europe
Jamaica                                                      Caribbean
Viet Nam                                            South-Eastern Asia
Romania                                                 Eastern Europe

                                                               DevName   1980  \
Country
India                                               Developing regions   8880
China                                               Developing regions   5123
United Kingdom of Great Britain and Northern Ir…   Developed regions  22045
Philippines                                         Developing regions   6051
Pakistan                                            Developing regions    978
United States of America                             Developed regions   9378
Iran (Islamic Republic of)                          Developing regions   1172
Sri Lanka                                           Developing regions    185
Republic of Korea                                   Developing regions   1011
Poland                                               Developed regions    863
Lebanon                                             Developing regions   1409
France                                               Developed regions   1729
Jamaica                                             Developing regions   3198
Viet Nam                                            Developing regions   1191
Romania                                              Developed regions    375

                                                     1981   1982   1983  \
Country
India                                                8670   8147   7338
China                                                6682   3308   1863
United Kingdom of Great Britain and Northern Ir…  24796  20620  10015
Philippines                                          5921   5249   4562
Pakistan                                              972   1201    900
United States of America                            10030   9074   7100
Iran (Islamic Republic of)                           1429   1822   1592
Sri Lanka                                             371    290    197
Republic of Korea                                    1456   1572   1081
Poland                                               2930   5881   4546
Lebanon                                              1119   1159    789
France                                               2027   2219   1490
Jamaica                                              2634   2661   2455
Viet Nam                                             1829   2162   3404
Romania                                               438    583    543
```

```
                                                   1984  1985  1986  …  \
Country                                                              …
India                                              5704  4211  7150  …
China                                              1527  1816  1960  …
United Kingdom of Great Britain and Northern Ir…  10170  9564  9470  …
Philippines                                        3801  3150  4166  …
Pakistan                                            668   514   691  …
United States of America                           6661  6543  7074  …
Iran (Islamic Republic of)                         1977  1648  1794  …
Sri Lanka                                          1086   845  1838  …
Republic of Korea                                   847   962  1208  …
Poland                                             3588  2819  4808  …
Lebanon                                            1253  1683  2576  …
France                                             1169  1177  1298  …
Jamaica                                            2508  2938  4649  …
Viet Nam                                           7583  5907  2741  …
Romania                                             524   604   656  …

                                                   2005   2006   2007  \
Country
India                                             36210  33848  28742
China                                             42584  33518  27642
United Kingdom of Great Britain and Northern Ir…   7258   7140   8216
Philippines                                       18139  18400  19837
Pakistan                                          14314  13127  10124
United States of America                           8394   9613   9463
Iran (Islamic Republic of)                         5837   7480   6974
Sri Lanka                                          4930   4714   4123
Republic of Korea                                  5832   6215   5920
Poland                                             1405   1263   1235
Lebanon                                            3709   3802   3467
France                                             4429   4002   4290
Jamaica                                            1945   1722   2141
Viet Nam                                           1852   3153   2574
Romania                                            5048   4468   3834

                                                   2008   2009   2010  \
Country
India                                             28261  29456  34235
China                                             30037  29622  30391
United Kingdom of Great Britain and Northern Ir…   8979   8876   8724
Philippines                                       24887  28573  38617
Pakistan                                           8994   7217   6811
United States of America                          10190   8995   8142
Iran (Islamic Republic of)                         6475   6580   7477
Sri Lanka                                          4756   4547   4422
Republic of Korea                                  7294   5874   5537
```

```
Poland                                               1267   1013    795
Lebanon                                              3566   3077   3432
France                                               4532   5051   4646
Jamaica                                              2334   2456   2321
Viet Nam                                             1784   2171   1942
Romania                                              2837   2076   1922

                                                     2011   2012   2013  \
Country
India                                               27509  30933  33087
China                                               28502  33024  34129
United Kingdom of Great Britain and Northern Ir…   6204   6195   5827
Philippines                                         36765  34315  29544
Pakistan                                             7468  11227  12603
United States of America                             7676   7891   8501
Iran (Islamic Republic of)                           7479   7534  11291
Sri Lanka                                            3309   3338   2394
Republic of Korea                                    4588   5316   4509
Poland                                                720    779    852
Lebanon                                              3072   1614   2172
France                                               4080   6280   5623
Jamaica                                              2059   2182   2479
Viet Nam                                             1723   1731   2112
Romania                                              1776   1588   1512

                                                     Total
Country
India                                               691904
China                                               659962
United Kingdom of Great Britain and Northern Ir…  551500
Philippines                                         511391
Pakistan                                            241600
United States of America                            241122
Iran (Islamic Republic of)                          175923
Sri Lanka                                           148358
Republic of Korea                                   142581
Poland                                              139241
Lebanon                                             115359
France                                              109091
Jamaica                                             106431
Viet Nam                                             97146
Romania                                              93585

[15 rows x 38 columns]
```

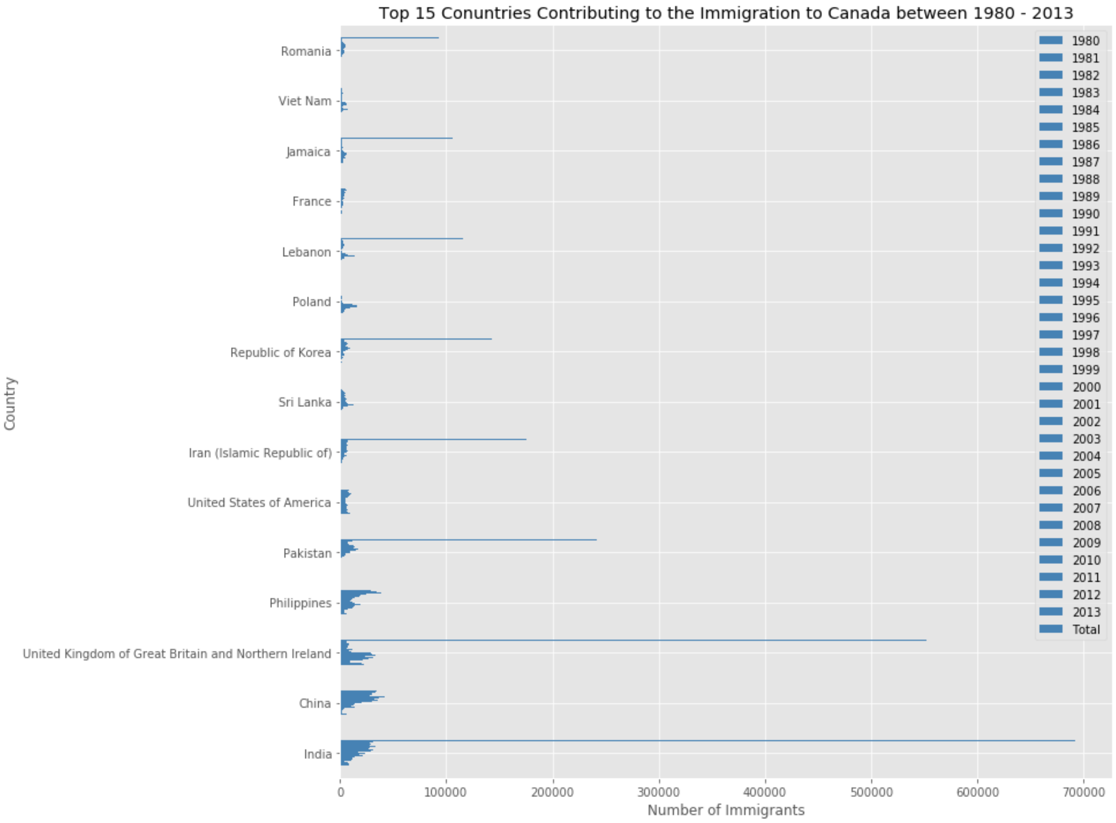
Double-click **here** for the solution.

Step 2: Plot data: 1. Use `kind='barh'` to generate a bar chart with horizontal bars. 2. Make sure to choose a good size for the plot and to label your axes and to give the plot a title. 3. Loop through the countries and annotate the immigrant population using the anotate function of the scripting interface.

[57]:

```
### type your answer here

df_top15.plot(kind='barh', figsize=(12, 12), color='steelblue')
plt.xlabel('Number of Immigrants')
plt.title('Top 15 Conuntries Contributing to the Immigration to Canada between␣
 ↪1980 - 2013')


for index, value in enumerate(df_top15):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 47000 from x, and 0.1 from y to␣
 ↪make it fit within the bar)
    plt.annotate(label, xy=(value - 47000, index - 0.10), color='white')

plt.show()
```

```
           ␣
↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
↪last)

        <ipython-input-57-9da6236d43ce> in <module>
          7
          8 for index, value in enumerate(df_top15):
    ----> 9     label = format(int(value), ',') # format int with commas
         10
         11     # place text at the end of bar (subtracting 47000 from x, and 0.
↪1 from y to make it fit within the bar)


        ValueError: invalid literal for int() with base 10: 'Continent'
```



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

6.0.1 Thank you for completing this lab!

This notebook was originally created by Jay Rajasekharan with contributions from Ehsan M. Kermani, and Slobodan Markovic.

This notebook was recently revamped by Alex Aklson. I hope you found this lab session interesting. Feel free to contact me if you have any questions!

This notebook is part of a course on **Coursera** called *Data Visualization with Python*. If you accessed this notebook outside the course, you can take this course online by clicking here.