Area Plots, Histograms, and Bar Charts

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Create and customize
  - Area plot
  - Histogram
  - Bar charts on a dataset

Table of Contents

Import Libraries

Import the `matplotlib` library.

```
#Import Primary Modules:
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library

# 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:  3.5.3
```

Fetching Data

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

In this lab, we will focus on the Canadian Immigration data and use the **already cleaned dataset** and can be fetched from here. You can refer to the lab on data pre-processing wherein this dataset is cleaned for a quick refresh your Panads skills Data pre-processing with Pandas

```
df_can = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-
SkillsNetwork/Data%20Files/Canada.csv')

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

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

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

```
df_can.head()
```

```
          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  ...  2005  2006  2007  2008  2009  2010
2011  \
0    39    47    71   340  ...  3436  3009  2652  2111  1746  1758
2203
1     0     0     0     0  ...  1223   856   702   560   716   561
539
2    71    69    63    44  ...  3626  4807  3623  4005  5393  4752
4325
3     0     0     0     0  ...     0     1     0     0     0     0
0
4     0     0     0     0  ...     0     1     1     0     0     0
0

   2012  2013  Total
0  2635  2004  58639
1   620   603  15699
2  3774  4331  69439
3     0     0      6
4     1     1     15

[5 rows x 39 columns]
```

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

```
# print the dimensions of the dataframe
print(df_can.shape)

(195, 39)
```

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

```
df_can.set_index('Country', inplace=True)

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

               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]
```

Notice now the country names now serve as indices.

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

data dimensions: (195, 38)

# 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

['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']
```

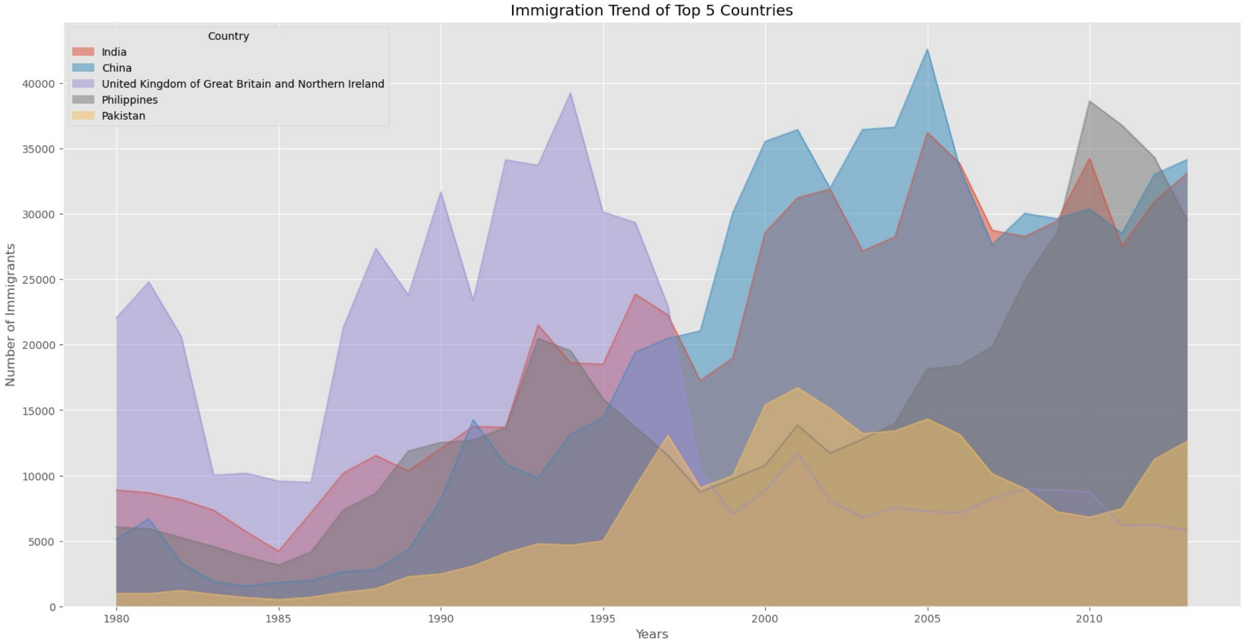
Area Plots

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**.

```
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()

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
```

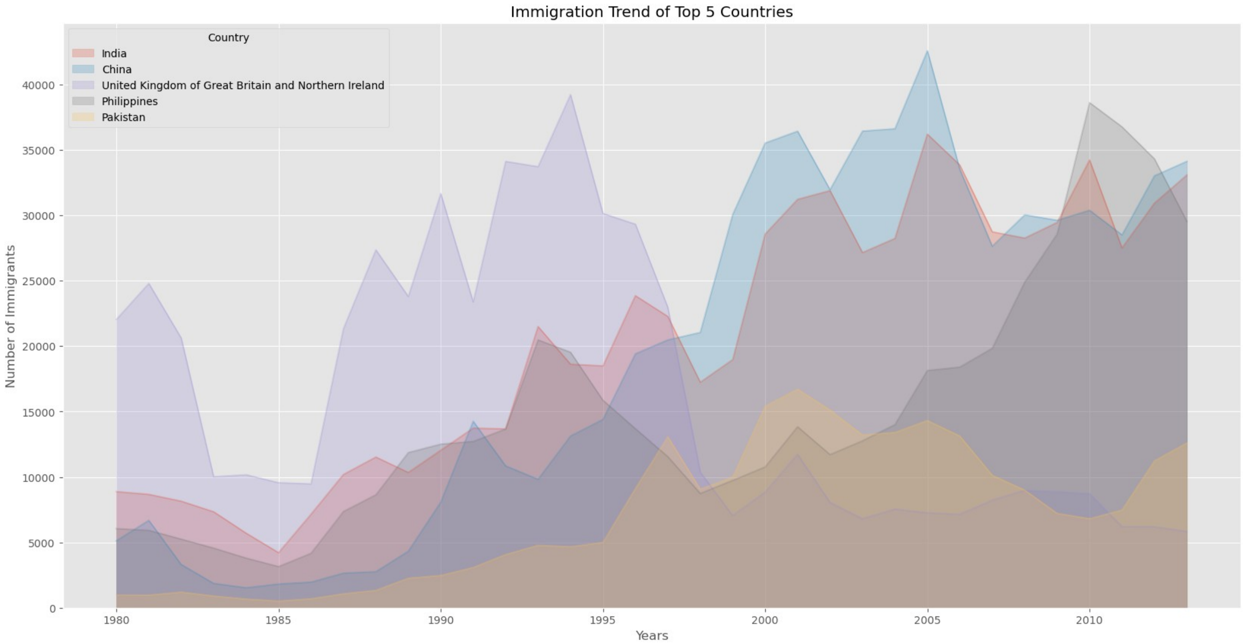
Area plots are stacked by default. And to produce a stacked area plot, each column must be either all positive or all negative values (any `NaN`, i.e. not a number, values will default to 0). To produce an unstacked plot, set parameter `stacked` to value `False`.

```
# let's change the index values of df_top5 to type integer for
plotting
df_top5.index = df_top5.index.map(int)
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.

```
df_top5.plot(kind='area',
             alpha=0.25,  # 0 - 1, default value alpha = 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()
```



Two types of plotting

As we discussed in the video lectures, there are two styles/options of plotting 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')
```

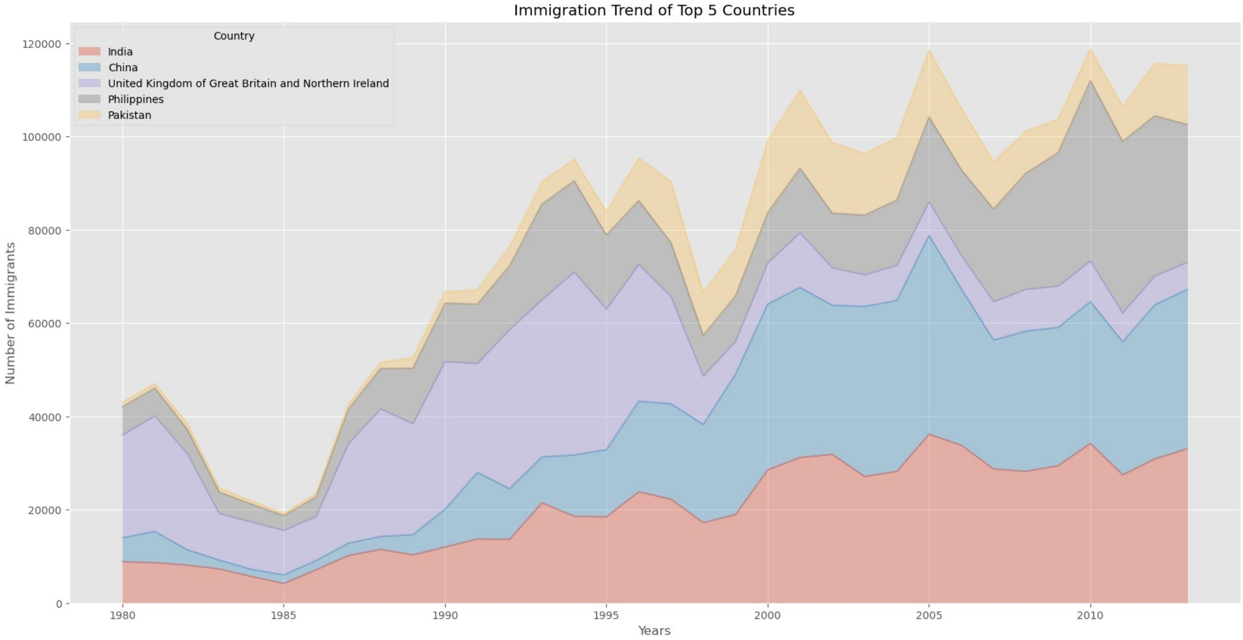
Option 2: Artist layer (Object oriented method) - using an `Axes` instance from Matplotlib (preferred)

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.

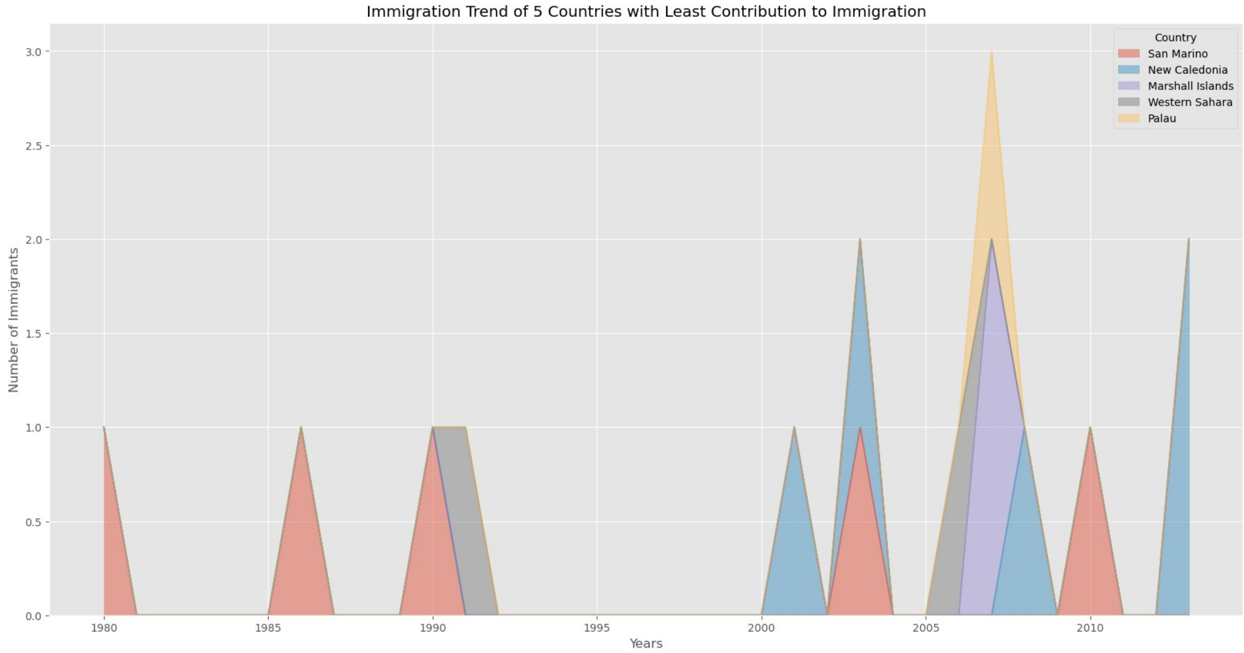
```
# option 2: preferred option with more flexibility
ax = df_top5.plot(kind='area', alpha=0.35, figsize=(20, 10))

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

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.

```
### type your answer here
df_least5 = df_can.tail(5)
df_least5 = df_least5[years].transpose()
df_least5.head()

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

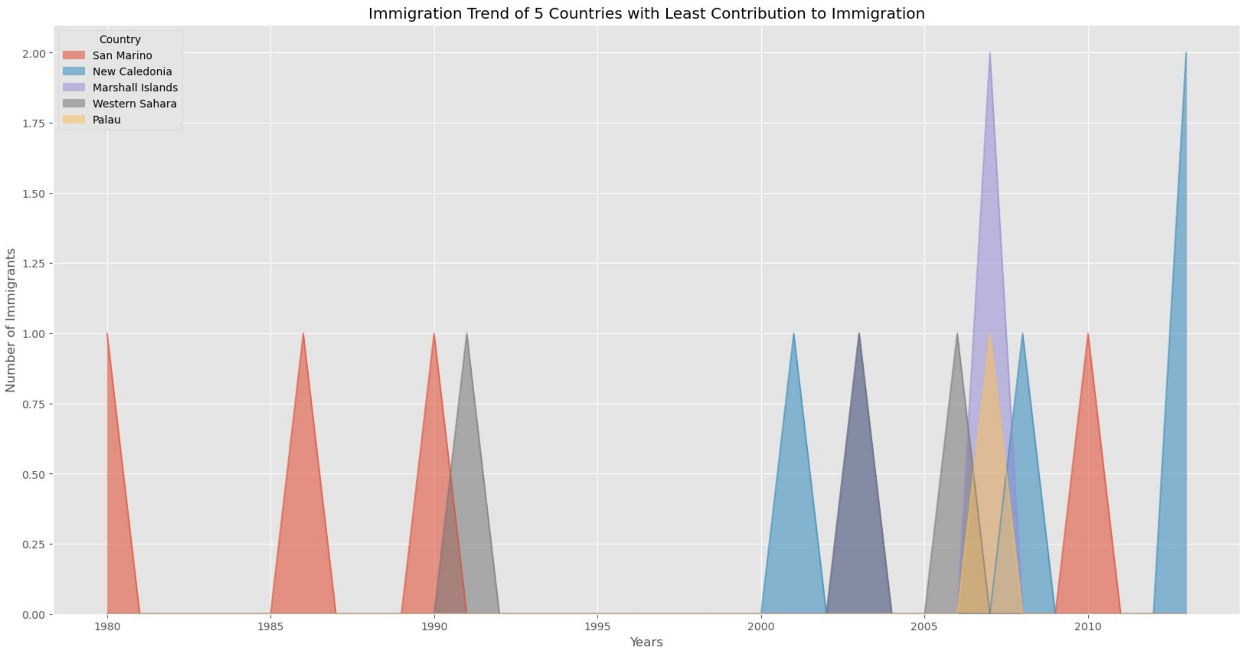
```
# get the 5 countries with the least contribution
df_least5 = df_can.tail(5)

# transpose the dataframe
df_least5 = df_least5[years].transpose()
df_least5.head()

df_least5.index = df_least5.index.map(int) # let's change the index
values of df_least5 to type integer for plotting
df_least5.plot(kind='area', alpha=0.45, figsize=(20, 10))

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

plt.show()
```



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.

```
# get the 5 countries with the least contribution
df_least5 = df_can.tail(5)

# transpose the dataframe
df_least5 = df_least5[years].transpose()

df_least5.head()
```

```
df_least5.index = df_least5.index.map(int) # let's change the index 
values of df_least5 to type integer for plotting

ax = df_least5.plot(kind='area', alpha=0.55, stacked=False, 
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')

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



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:

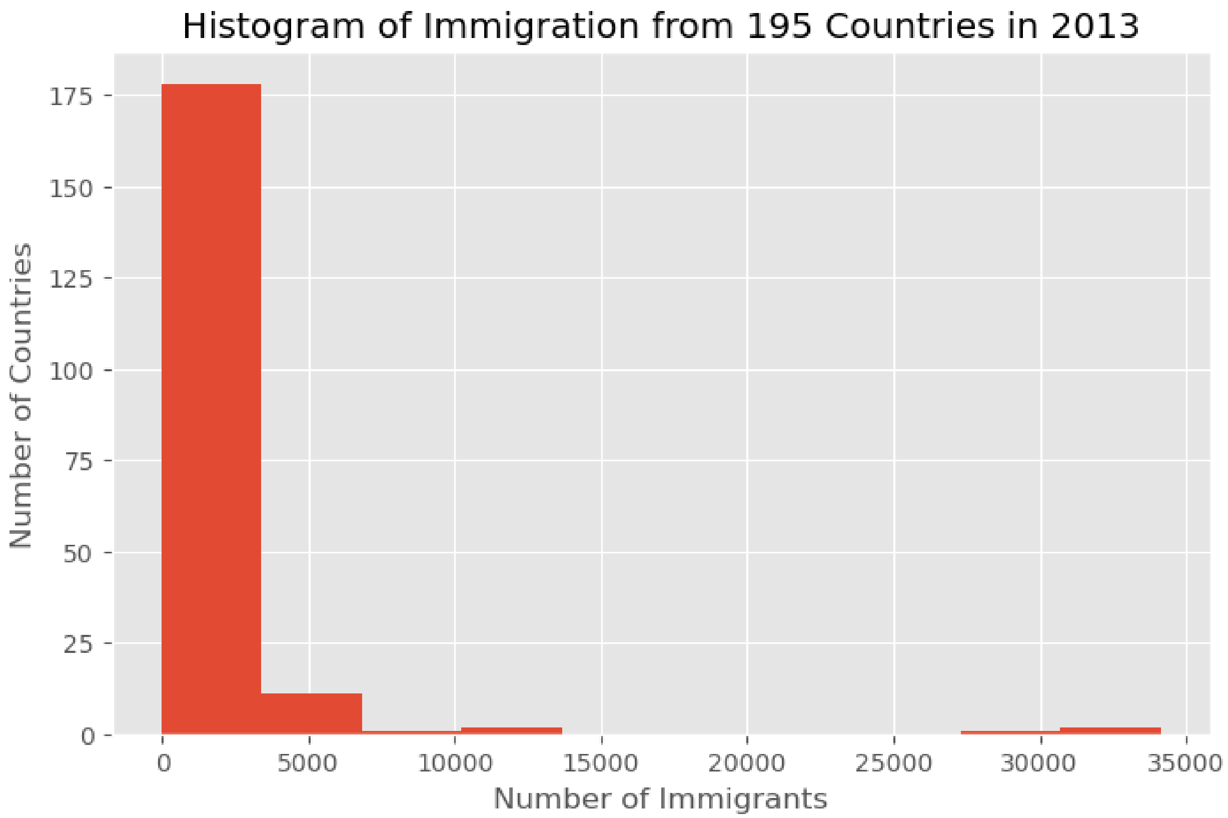
```
# let's quickly view the 2013 data
df_can['2013'].head()

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

# 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()`.

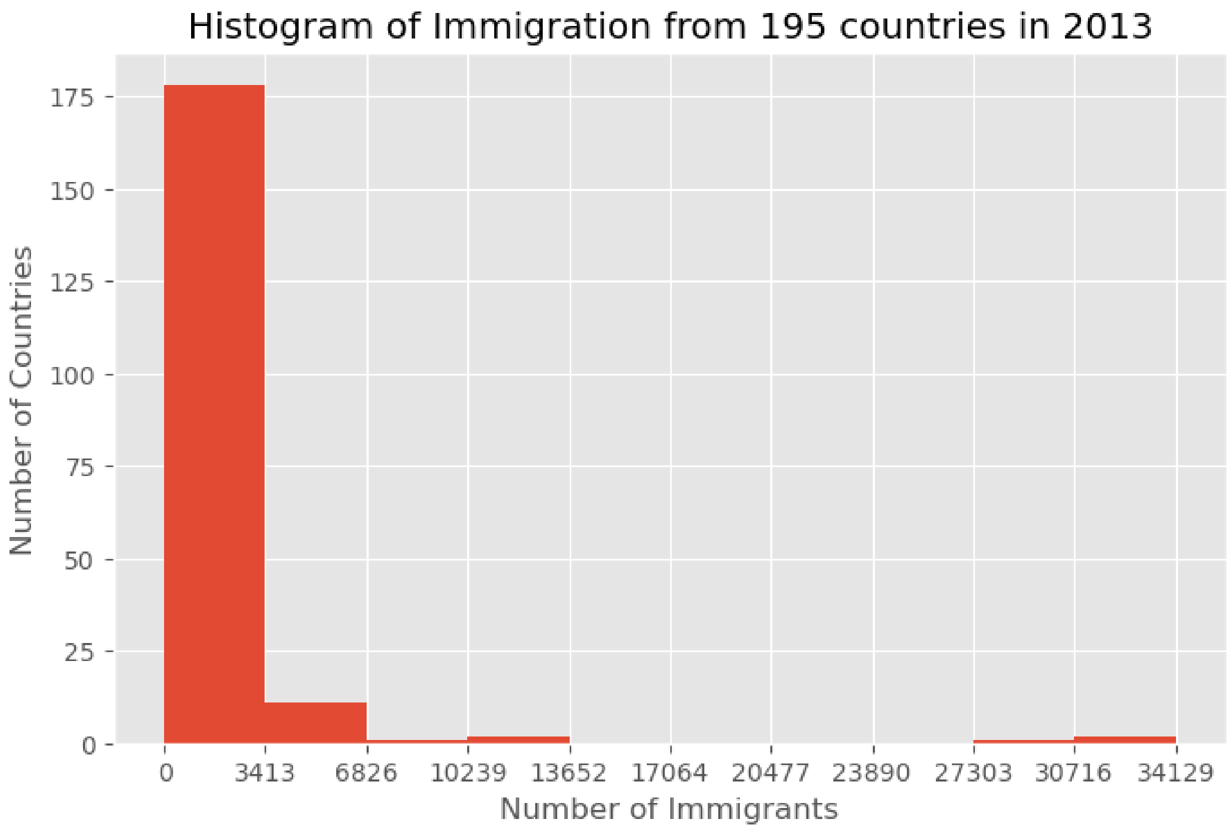
```
df_can['2013'].plot(kind='hist', figsize=(8, 5))

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

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:

```
# '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?

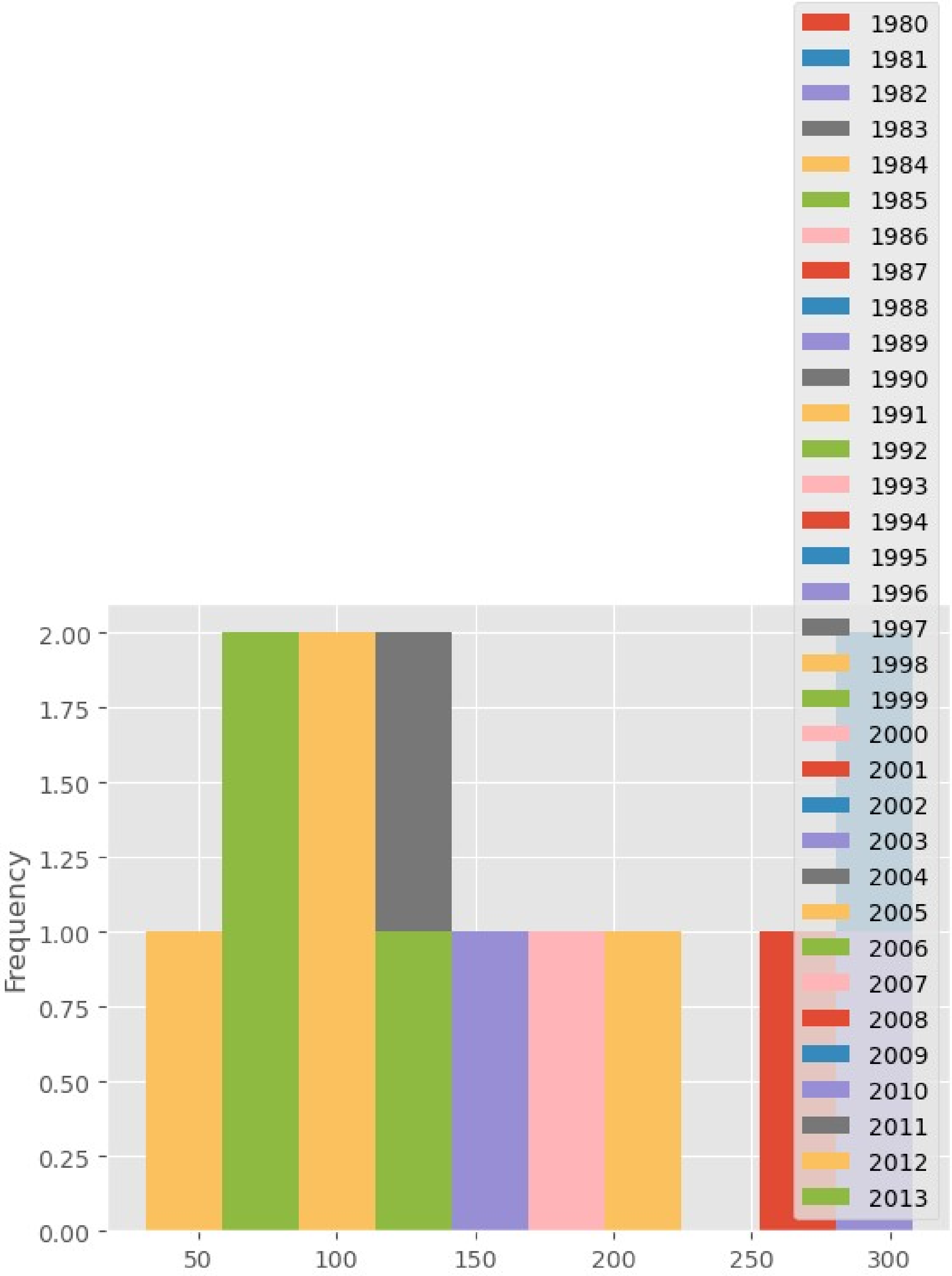
```
# let's quickly view the dataset
df_can.loc[['Denmark', 'Norway', 'Sweden'], years]
```

```
         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]
```

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

```
<AxesSubplot:ylabel='Frequency'>
```



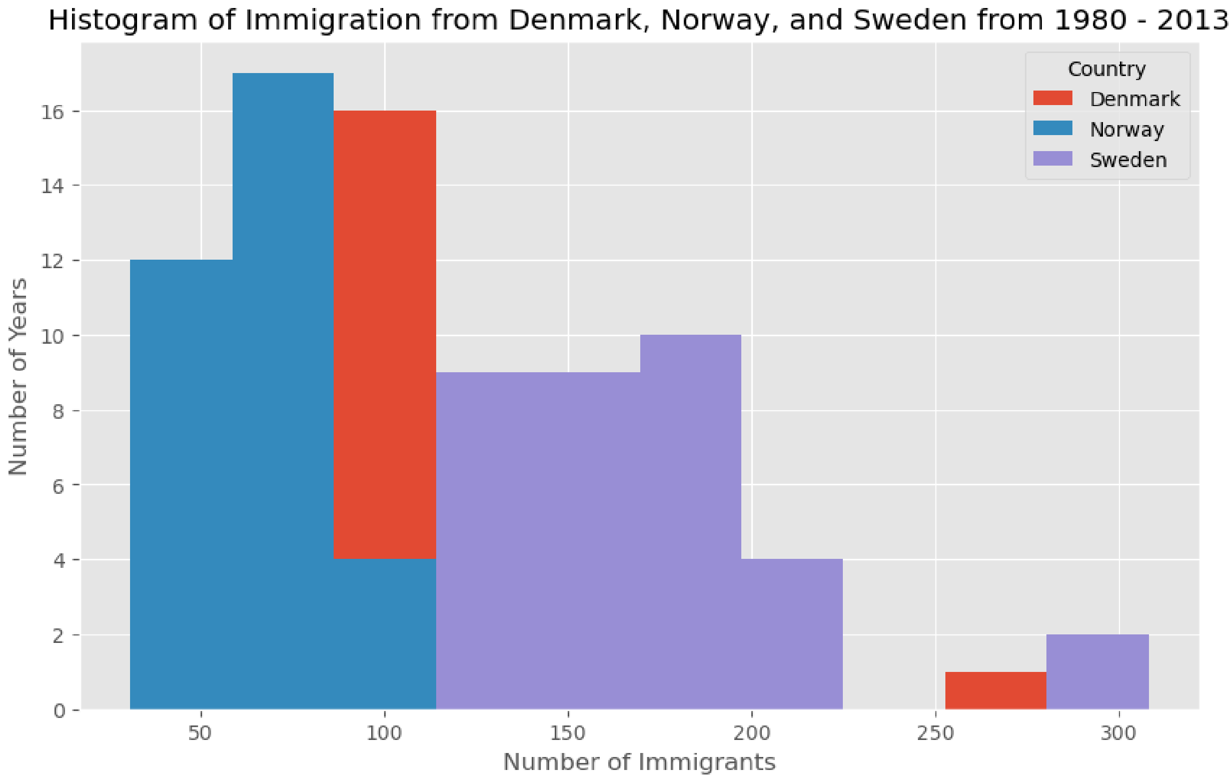
That does not look right!

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.

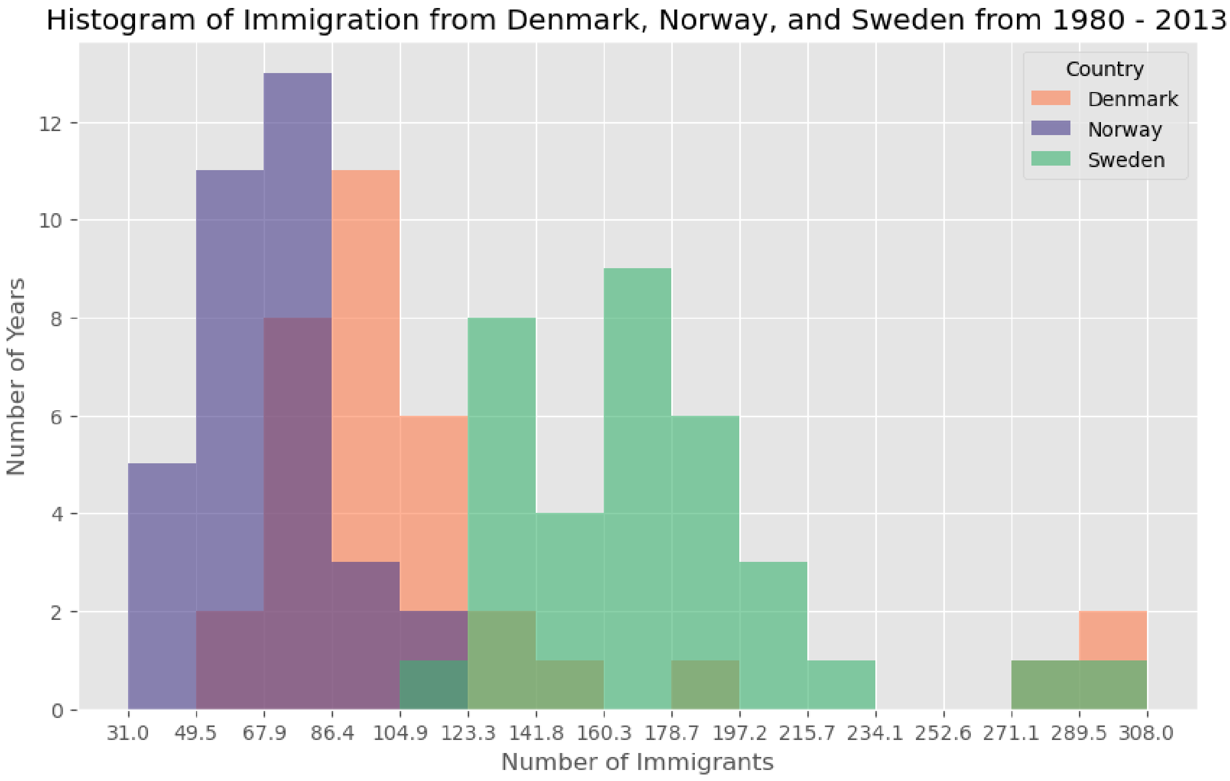
```
# transpose dataframe
df_t = df_can.loc[['Denmark', 'Norway', 'Sweden'], years].transpose()
df_t.head(5)

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

# 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` parameter;
- label the x-axis by passing in `x-label` parameter;
- change the colors of the plots by passing in `color` parameter.

```
# 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()
```

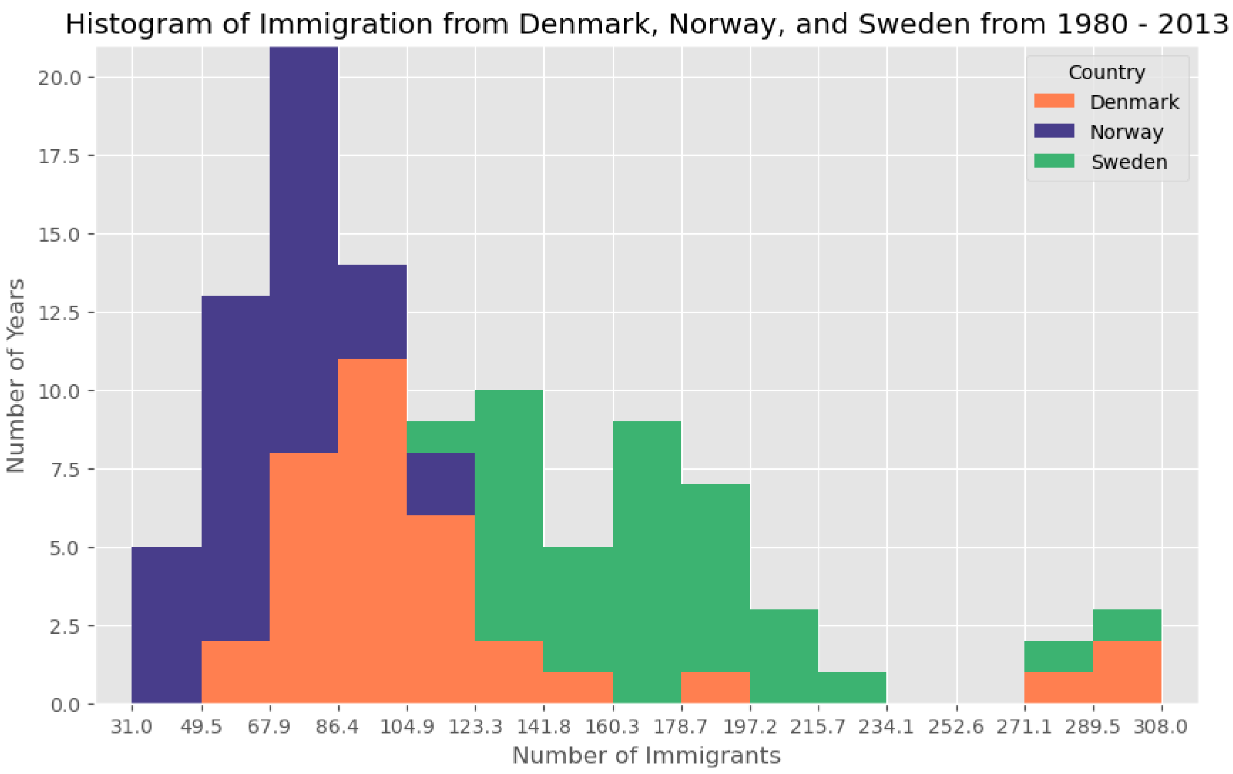


Tip: For a full listing of colors available in Matplotlib, run the following code in your python shell:

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

If we do not want the plots to overlap each other, we can stack them using the `stacked` parameter. 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.

```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()
```



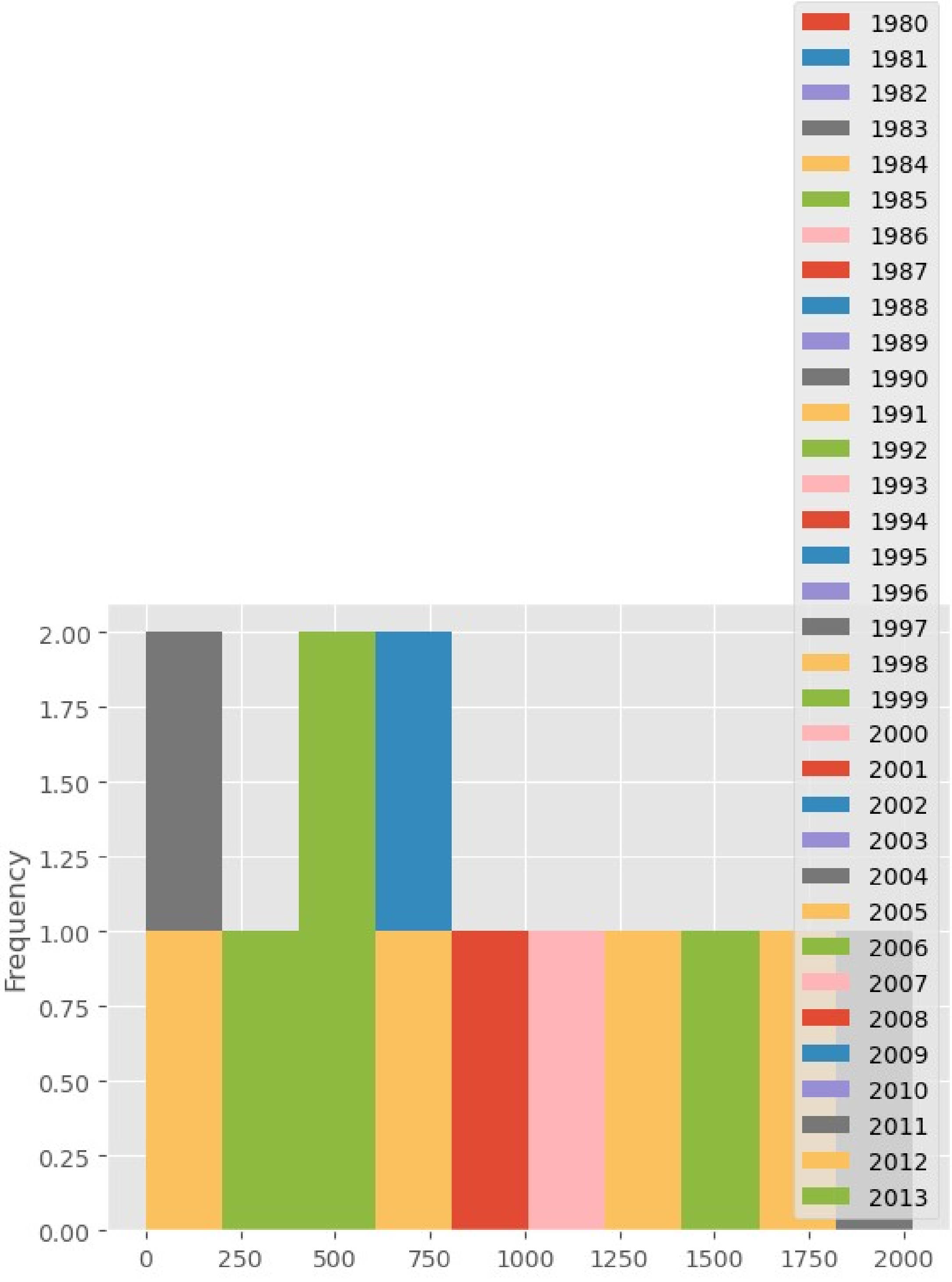
Question: Use the scripting layer to display the immigration distribution for Greece, Albania, and Bulgaria for years 1980 - 2013? Use an overlapping plot with 15 bins and a transparency value of 0.35.

```
### type your answer here
# let's quickly view the dataset
df_can.loc[['Greece', 'Albania', 'Bulgaria'], years]
```

```
          1980  1981  1982  1983  1984  1985  1986  1987  1988
1989  ...  \
Country
...
Greece    1065   953   897   633   580   584   547   765   564
773  ...
Albania      1     0     0     0     0     0     1     2     2
3  ...
Bulgaria    24    20    12    33    11    24    33    52    43
85  ...

          2004  2005  2006  2007  2008  2009  2010  2011  2012  2013
Country
Greece     120   100    74   110   107   119   101   102   146   298
Albania   1450  1223   856   702   560   716   561   539   620   603
Bulgaria  2022  1738  1419  1172   994   784   556   365   451   512

[3 rows x 34 columns]
```

```
# generate histogram
df_can.loc[['Greece', 'Albania', 'Bulgaria'], years].plot.hist()
```

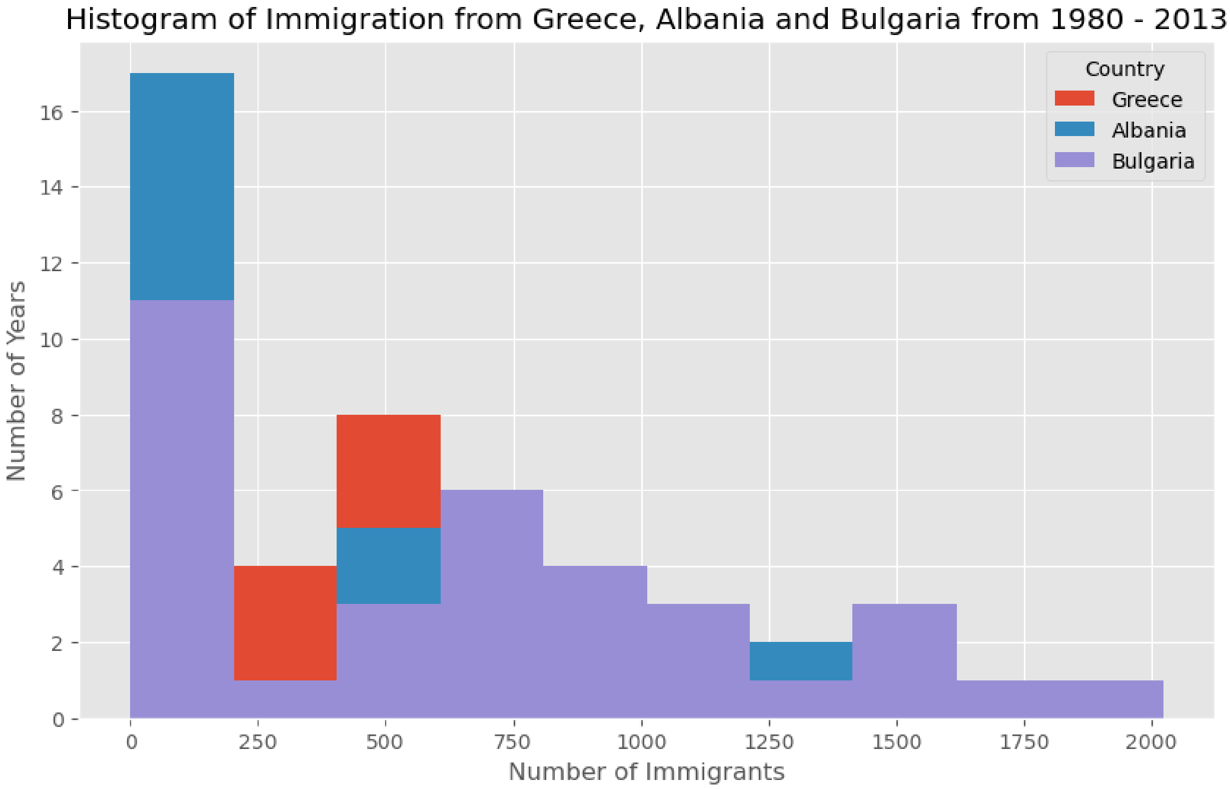
```
<AxesSubplot:ylabel='Frequency'>
```



```
# transpose dataframe
df_t = df_can.loc[['Greece', 'Albania', 'Bulgaria'],
years].transpose()
df_t.head(5)
```

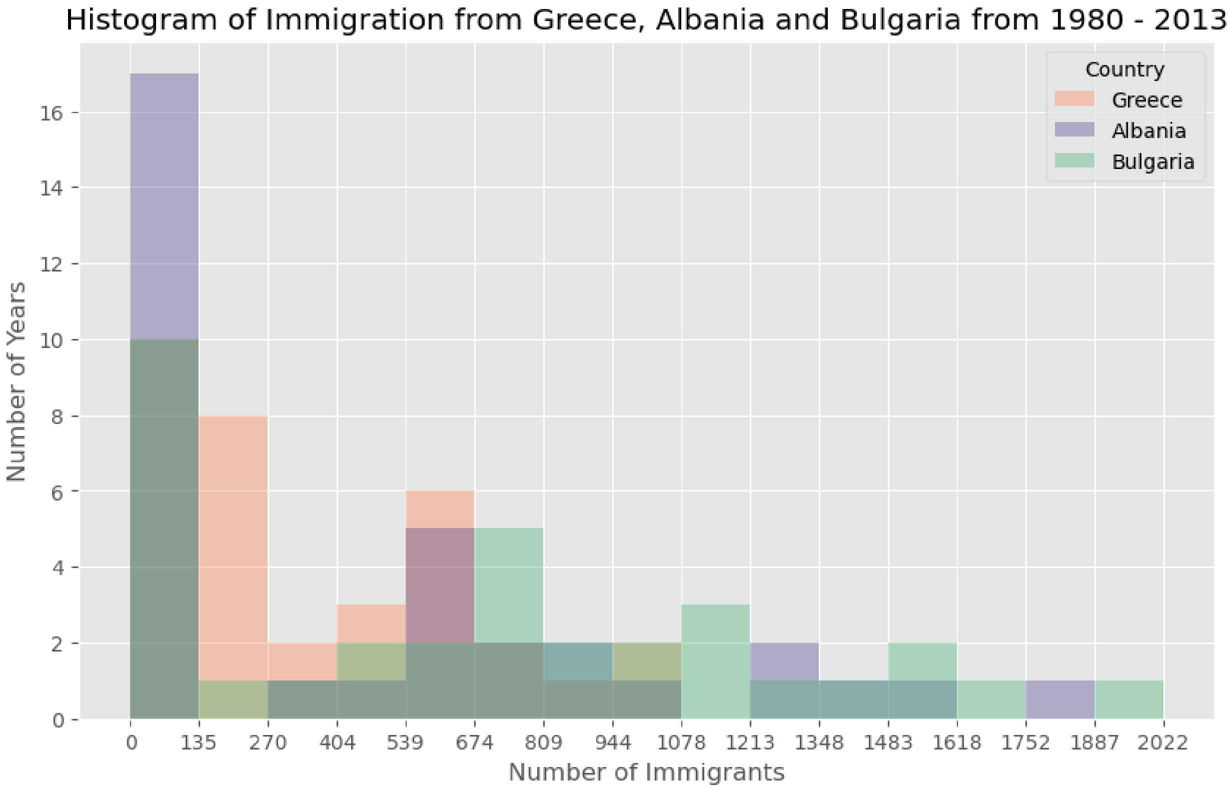
```
Country  Greece  Albania  Bulgaria
1980       1065        1        24
1981        953        0        20
1982        897        0        12
1983        633        0        33
1984        580        0        11
```

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

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()
```



```
# 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=0.35,
          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()
```



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 particularly 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:

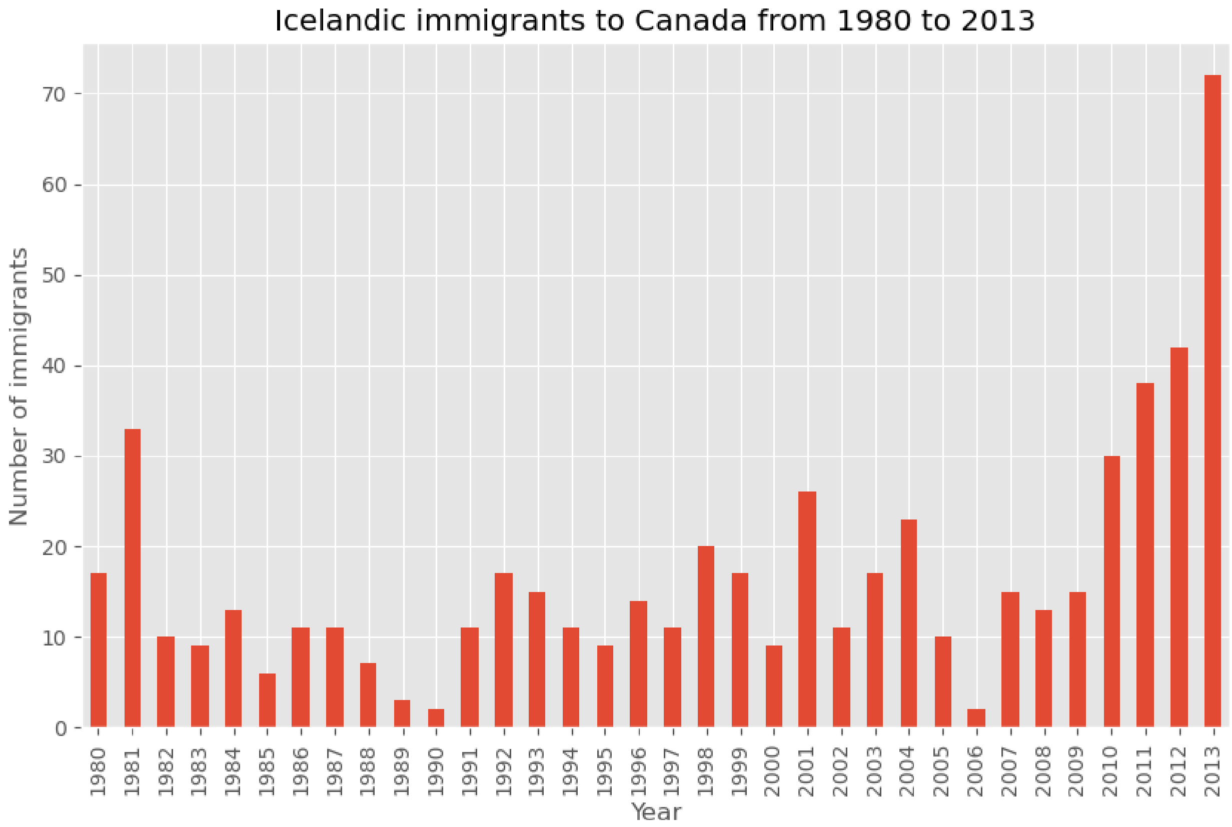
The 2008 - 2011 Icelandic Financial Crisis was a major economic and political event in Iceland. Relative to the size of its economy, Iceland's systemic banking collapse was the largest experienced by any country in economic history. The crisis led to a severe economic depression in 2008 - 2011 and significant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

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

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

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

```
# 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()
```



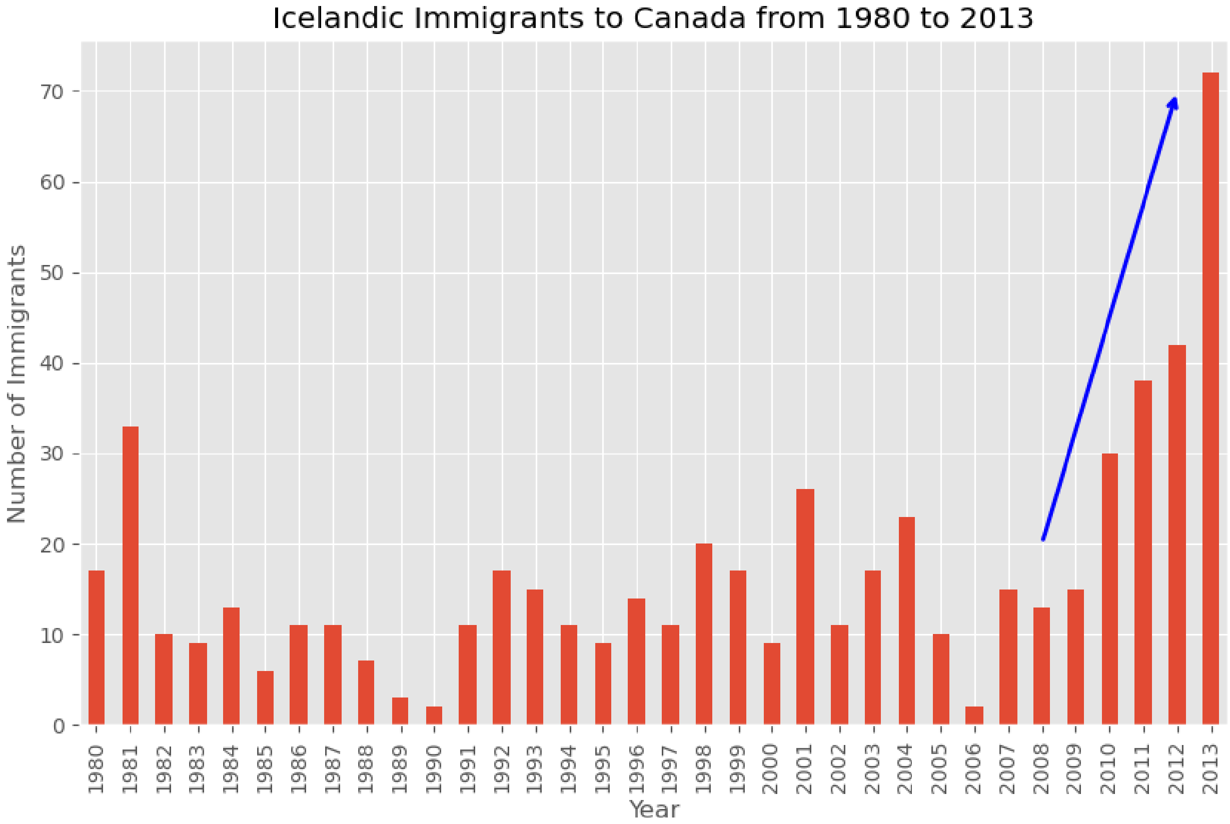
The bar plot above shows the total number of immigrants broken down by each year. We can clearly see the impact of the financial crisis; the number of immigrants to Canada started increasing rapidly after 2008.

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`: Specifies color of arrow.
  - `lw`: Specifies the line width.

I encourage you to read the Matplotlib documentation for more details on annotations: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.annotate.html.

```
df_iceland.plot(kind='bar', figsize=(10, 6), rot=90)  # rotate the
xticks(labelled points on x-axis) 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()
```

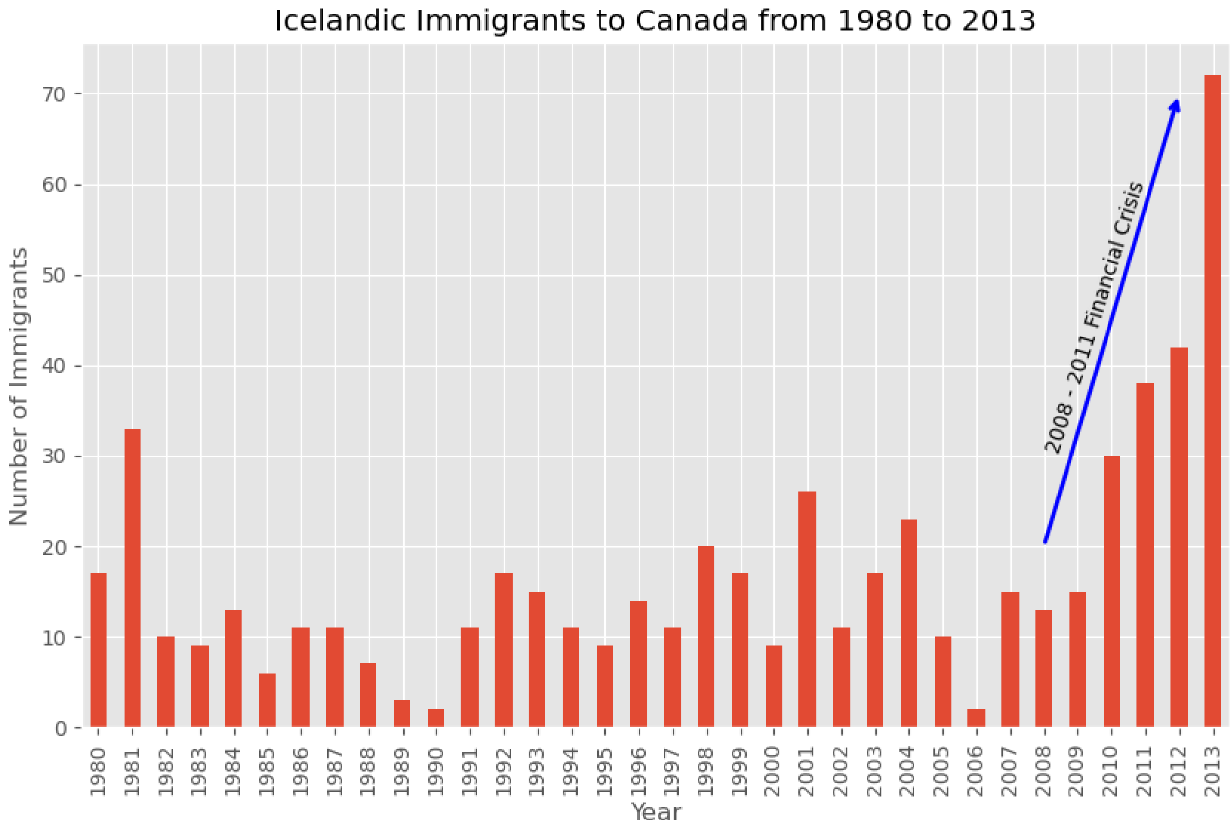


Let's also annotate a text to go over the arrow. We will pass in the following additional parameters:

- `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’]

```
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 categorical variables.

Question: Using the scripting later 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.

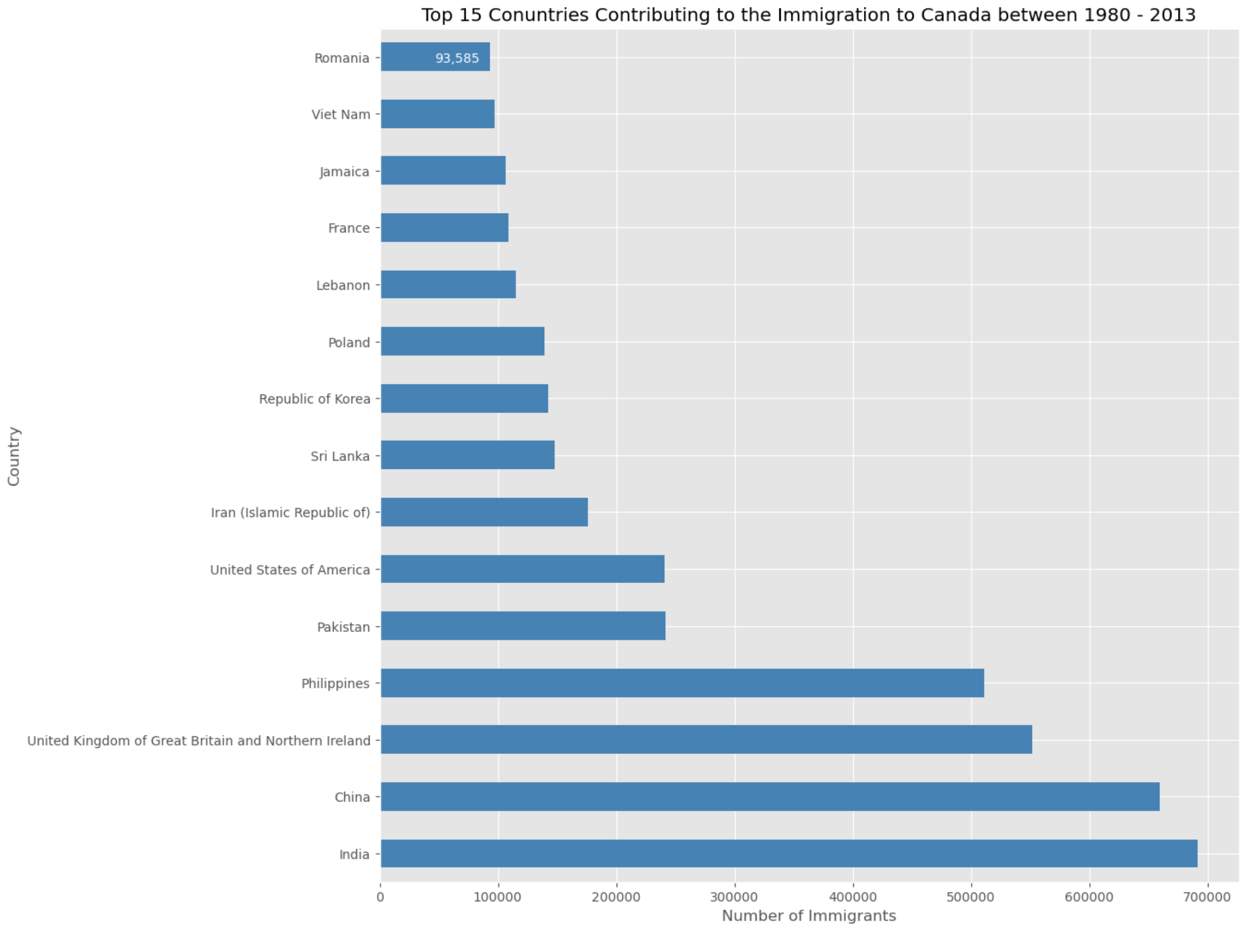
```
# sort dataframe on 'Total' column (descending)
df_can.sort_values(by='Total', ascending=True, inplace=True)

# get top 15 countries
df_top15 = df_can['Total'].tail(15)
df_top15

Country
Romania                                                      93585
Viet Nam                                                     97146
Jamaica                                                     106431
France                                                      109091
```

```
Lebanon                                                 115359
Poland                                                  139241
Republic of Korea                                       142581
Sri Lanka                                               148358
Iran (Islamic Republic of)                              175923
United States of America                                241122
Pakistan                                                241600
Philippines                                             511391
United Kingdom of Great Britain and Northern Ireland    551500
China                                                   659962
India                                                   691904
Name: Total, dtype: int64
```

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.

```
# generate plot
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')

# annotate value labels to each country
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()
```



Thank you for completing this lab!

Author

Alex Aklson

Other Contributors

Jay Rajasekharan, Ehsan M. Kermani, Slobodan Markovic, Weiqing Wang, Pooja.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-06-11	2.5	Pooja	Changed to work on clean data, links updated
2021-05-29	2.4	Weiqing Wang	Fixed typos and code spells.
2021-01-20	2.3	Lakshmi Holla	Changed TOC cell markdown
2021-01-05	2.2	Lakshmi Holla	Changed solution code for annotate
2020-11-03	2.1	Lakshmi Holla	Changed the URL of excel file

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab