**Area Plots, Histograms, and Bar Plots**

Estimated time needed: **30** minutes

**Objectives**

After completing this lab you will be able to:

* Create additional labs namely area plots, histogram and bar charts

**Table of Contents**

1. [Exploring Datasets with *pandas*](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-Exercise-Area-Plots-Histograms-and-Bar-Charts.ipynb#0)
2. [Downloading and Prepping Data](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-Exercise-Area-Plots-Histograms-and-Bar-Charts.ipynb#2)
3. [Visualizing Data using Matplotlib](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-Exercise-Area-Plots-Histograms-and-Bar-Charts.ipynb#4)
4. [Area Plots](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-Exercise-Area-Plots-Histograms-and-Bar-Charts.ipynb#6)
5. [Histograms](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-Exercise-Area-Plots-Histograms-and-Bar-Charts.ipynb#8)
6. [Bar Charts](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-Exercise-Area-Plots-Histograms-and-Bar-Charts.ipynb#10)

**Exploring Datasets with *pandas* and Matplotlib**

Toolkits: The course heavily relies on [**pandas**](http://pandas.pydata.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) and **[Numpy](http://www.numpy.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01" \t "_blank)** for data wrangling, analysis, and visualization. The primary plotting library that we are exploring in the course is [Matplotlib](http://matplotlib.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01" \t "_blank).

Dataset: Immigration to Canada from 1980 to 2013 - [International migration flows to and from selected countries - The 2015 revision](http://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.shtml?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) 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.

**Downloading and Prepping Data**

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

[ ]:



**import** numpy **as** np *# useful for many scientific computing in Python*

**import** pandas **as** pd *# primary data structure library*

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

! pip3 install openpyxl

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

[ ]:



df\_can **=** pd.read\_excel(

'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.xlsx',

sheet\_name**=**'Canada by Citizenship',

skiprows**=**range(20),

skipfooter**=**2)

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

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

[ ]:



df\_can.head()

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

[ ]:



*# print the dimensions of the dataframe*

print(df\_can.shape)

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).**

[ ]:



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()

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.**

[ ]:



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()

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.**

[ ]:



*# let's examine the types of the column labels*

all(isinstance(column, str) **for** column **in** df\_can.columns)

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.

[ ]:



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)

**4. 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()

Notice now the country names now serve as indices.

**5. Add total column.**

[ ]:



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()

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:

[ ]:



print('data dimensions:', df\_can.shape)

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

[ ]:



*# 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

**Visualizing Data using Matplotlib**

Import the matplotlib 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*

**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()

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')

**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*

​

​

​

​

Click here for a sample python solution

*#The correct answer is:*

*# 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.

[ ]:



*### type your answer here*

​

​

​

​

Click here for a sample python solution

*#The correct answer is:*

*# 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')

**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()

[ ]:



*# 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*

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](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.plot.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

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]

[ ]:



*# generate histogram*

df\_can.loc[['Denmark', 'Norway', 'Sweden'], years].plot.hist()

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()

[ ]:



*# 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:

**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.

[ ]:



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*

​

​

​

​

Click here for a sample python solution

*#The correct answer is:*

*# create a dataframe of the countries of interest (cof)*

df\_cof **=** df\_can.loc[['Greece', 'Albania', 'Bulgaria'], years]

*# transpose the dataframe*

df\_cof **=** df\_cof.transpose()

*# let's get the x-tick values*

count, bin\_edges **=** np.histogram(df\_cof, 15)

*# Un-stacked Histogram*

df\_cof.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()

[ ]:



*# 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: <http://matplotlib.orsg/api/pyplot_api.html#matplotlib.pyplot.annotate>.

[ ]:



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.

[ ]:



*### type your answer here*

​

​

​

​

Click here for a sample python solution

*#The correct answer is:*

*# 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

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.

[ ]:



*### type your answer here*

​

​

​

​

Click here for a sample python solution

*#The correct answer is:*

*# 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**