Data Visualization

PERTEMUAN XV & XVI

Introduction to Visualization Tools

Why Build Visuals?

- Data visualization is a way to show a complex data in a form that is graphical and easy to understand
- This can be especially useful when one is trying to explore the data
- A picture is worth a thousand words, then plots and graphs can be very effective in communicate data clearly
- It can be very valuable when it comes to supporting any recommendations to different stakeholders

Best Practices

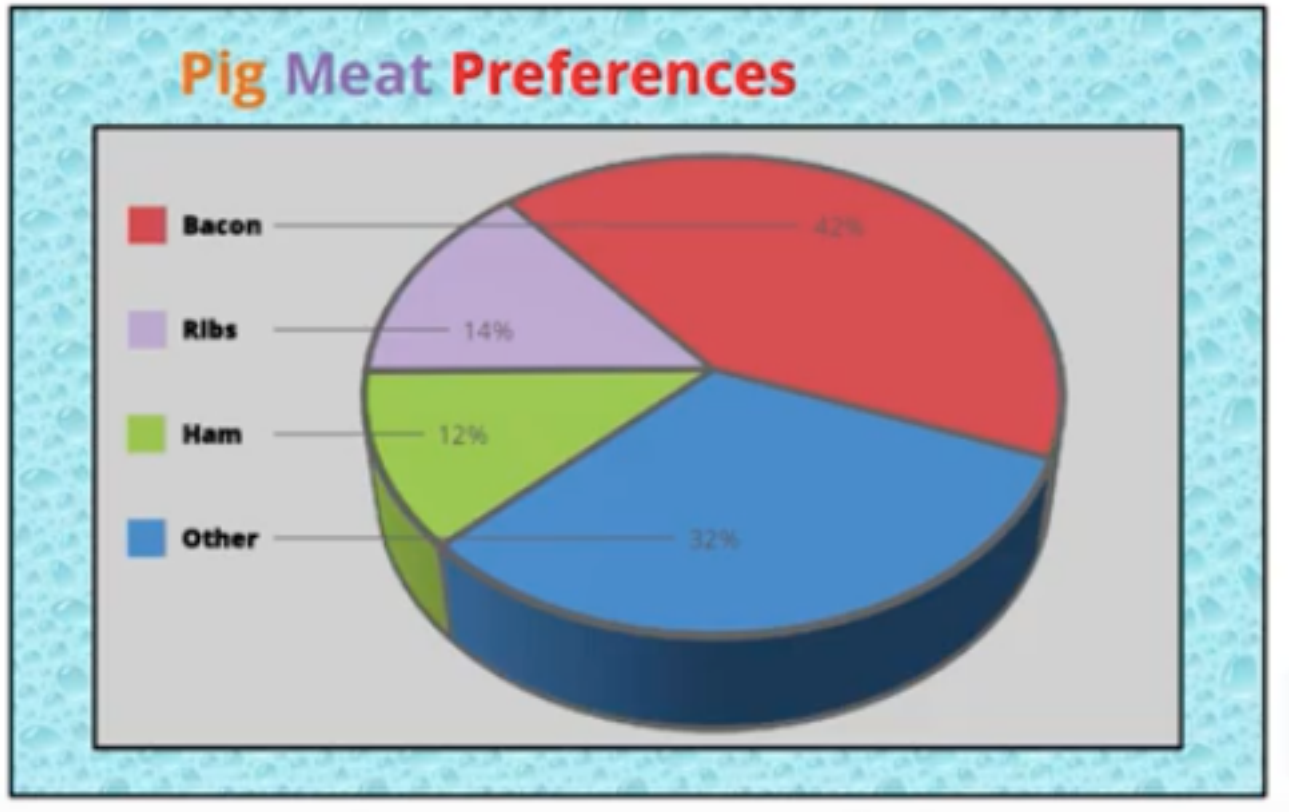
When creating a visual, always remember :

1. Less is more effective
2. Less is more attractive
3. Less is more impactful

In other words, any feature or design you incorporate in your plot to make it more attractive or pleasing should support the message that the plot is meant to get across and not distract from it

Best Practices

- Let's look at an example. We're not even sure it features such as the blue background or 3D orientation are meant to convey anything.
- In fact, these additional unnecessary features distract from the main message and can be confusing to the audience



Best Practices

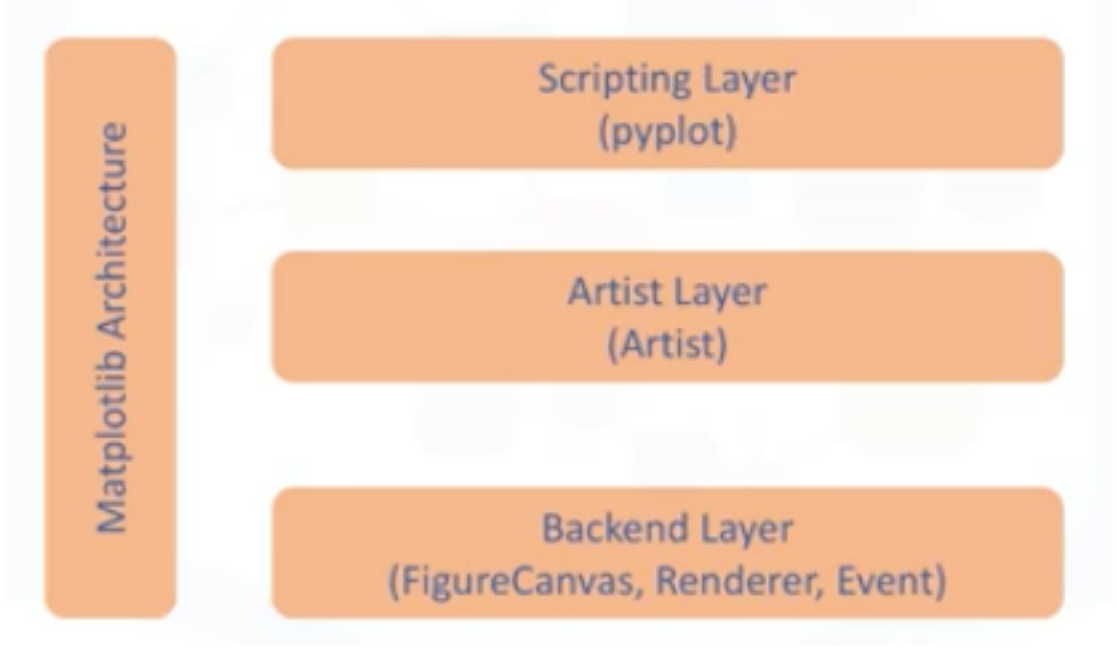
The message here is that people are most likely to choose bacon over other types of pig meat, so let's get rid of everything that can be distracting from this core message.

It is simple, cleaner, less distracting, and much easier to read.

Matplotlib

Matplotlib is one of the most widely used data visualization library in Python.

Matplotlib's architecture is composed of three main layers: the back-end layer, the artist layer, and the scripting layer.



Matplotlib – Back-end Layer

The back-end layer has three built-in abstract interface classes:

-FigureCanvas, which defines and encompasses the area on which the figure is drawn.

matplotlib.backend_bases.FigureCanvas

-Renderer, an instance of the renderer class knows how to draw on the figure canvas.

matplotlib.backend_bases.Renderer

-Event, which handles user inputs such as keyboard strokes and mouse clicks.

matplotlib.backend_bases.Event

Matplotlib – Artist Layer

- Artist Layer is composed of one main object, which is the Artist.
- The Artist is the object that knows how to take the Renderer and use it to put ink on the canvas.
- Everything you see on a Matplotlib figure is an Artist instance. The title, the lines, the tick labels, the images, and so on, all correspond to an individual Artist.
- 2 Types of Artist object :

1. Primitive : Line2D, Rectangle, Circle, and Text
2. Composite : Axis, Tick, Axes, and Figure

Each composite artist may contain other composite artists as well as primitive artists.

Matplotlib – Artist Layer

Let’s try to generate a histogram of 10,000 random numbers using the artist layer.

First, we import the figure canvas from the backend backend underscore agg and attach the figure artist to it. Agg stands for anti grain geometry which is a high-performance library that produces attractive images

```
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas # import FigureCanvas
from matplotlib.figure import Figure # import Figure artist
fig = Figure()
canvas = FigureCanvas(fig)
```

Matplotlib – Artist Layer

Then we import the Numpy library to generate the random numbers.

```
# create 10000 random numbers using numpy
import numpy as np
x = np.random.randn(10000)
```

Next, we create an axes artist. The axes artist is added automatically to the figure axes container, Fig.axes.

```
ax = fig.add_subplot(111) # create an axes artist
```

And note here that (111) is from the MATLAB convention so it creates a grid with one row and one column and uses the first cell in that grid for the location of the new axes.

Matplotlib – Artist Layer

Then we call the axes method hist, to generate the histogram. hist creates a sequence of rectangle artists for each histogram bar and adds them to the axes container. Here 100 means create 100 bins

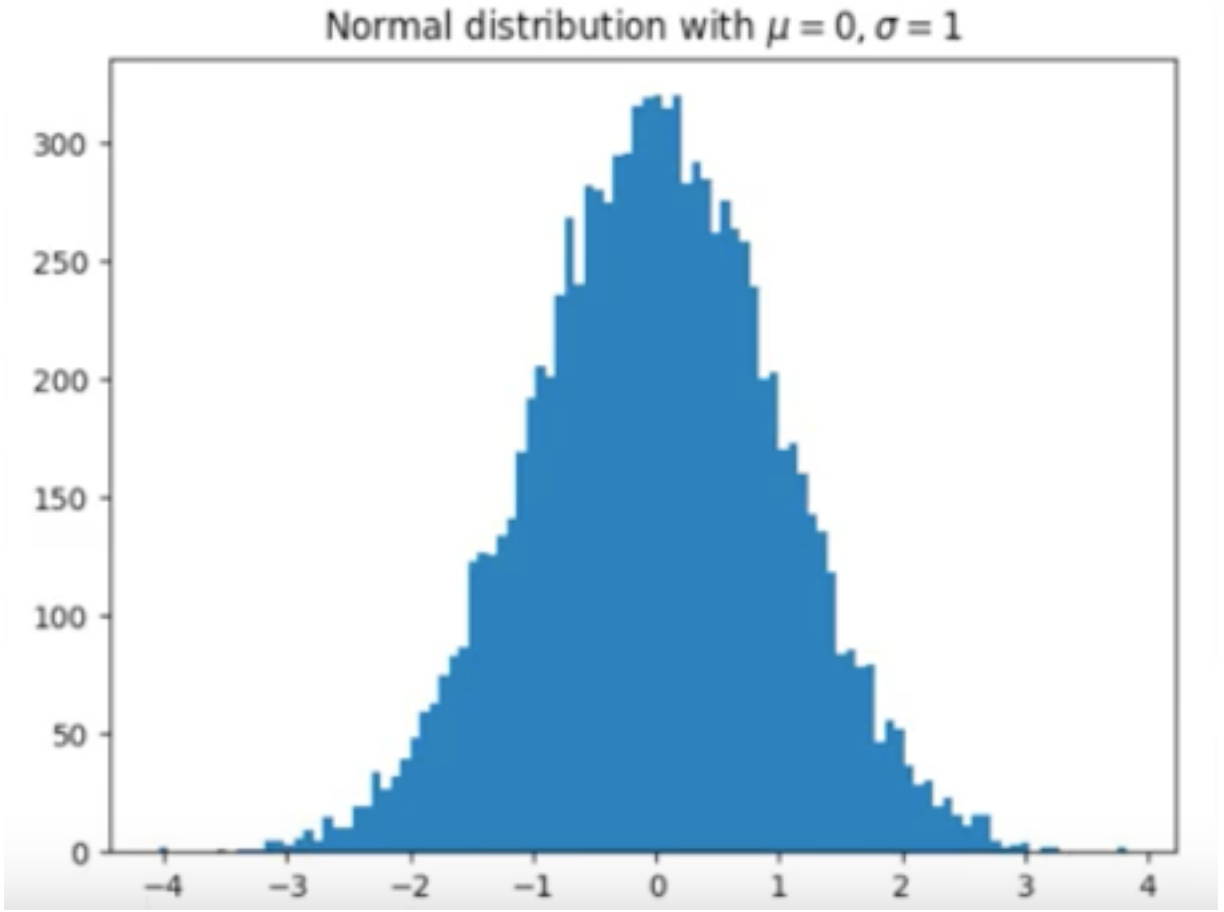
```
ax.hist(x, 100) # generate a histogram of the 10000 numbers
```

Finally we decorate the figure with a title, and we save it.

```
# add a title to the figure and save it
ax.set_title('Normal distribution with $\mu=0, \sigma=1$')
fig.savefig('matplotlib_histogram.png')
```

Matplotlib – Artist Layer

Now this is the generated histogram and so this is how we use the artist layer to generate a graphic.



Matplotlib – Scripting Layer

- As for the scripting layer, it was developed for scientists who are not professional programmers
- Histogram that we just created before in artist layer is syntactically heavy as it is meant for developers and not for individuals whose goal is to perform quick exploratory analysis of some data.
- Matplotlib's scripting layer is essentially the Matplotlib.pyplot interface, a scripting interface that is lighter than the Artist layer
- Let's see how we can generate a histogram of 10,000 random numbers using the pyplot interface
- So first we import the pyplot interface and you can see how all the methods associated with creating the histogram and other artist objects and manipulating them whether it is the hist method or showing the figure are part of the pyplot interface.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randn(10000)
plt.hist(x, 100)
plt.title(r'Normal distribution with $\mu=0, \sigma=1$')
plt.savefig('matplotlib_histogram.png')
plt.show()
```

Basic Plotting with Matplotlib

- We will learn how to use Matplotlib to create plots and we are using the Jupyter notebook as our environment.
- If you start a Jupyter notebook, all you have to do is import Matplotlib and you're ready to go.

Matplotlib – Plot Function

- We can create almost all the conventional visualization tools such as histogram, bar charts, etc using plot function.
- Let's start with an example. Let's first import the scripting interface as plt and let's plot a circular mark at the position (5, 5)

Matplotlib - Pandas

- Plotting in pandas is as simple as calling the plot function on a given pandas series or dataframe.
- So, say we have a data frame of the number of immigrants from India and China to Canada from 1980 to 1996 and say we're interested in generating a line plot of these data.

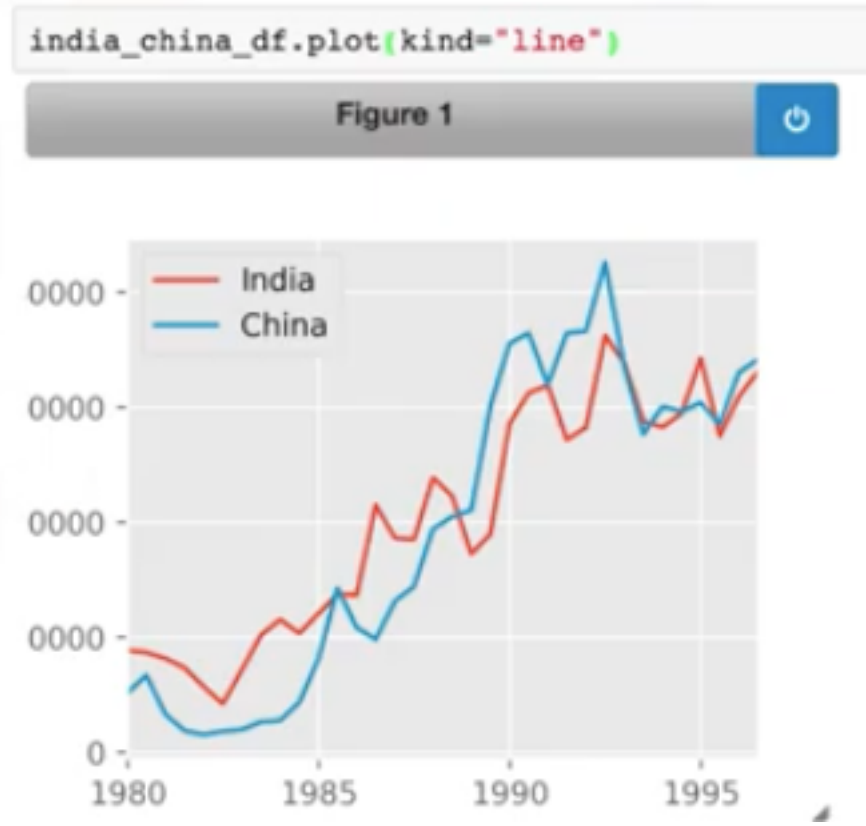
india_china_df

	India	China
1980	8880	5123
1981	8670	6682
1982	8147	3308
1983	7338	1863
1984	5704	1527

Matplotlib - Pandas

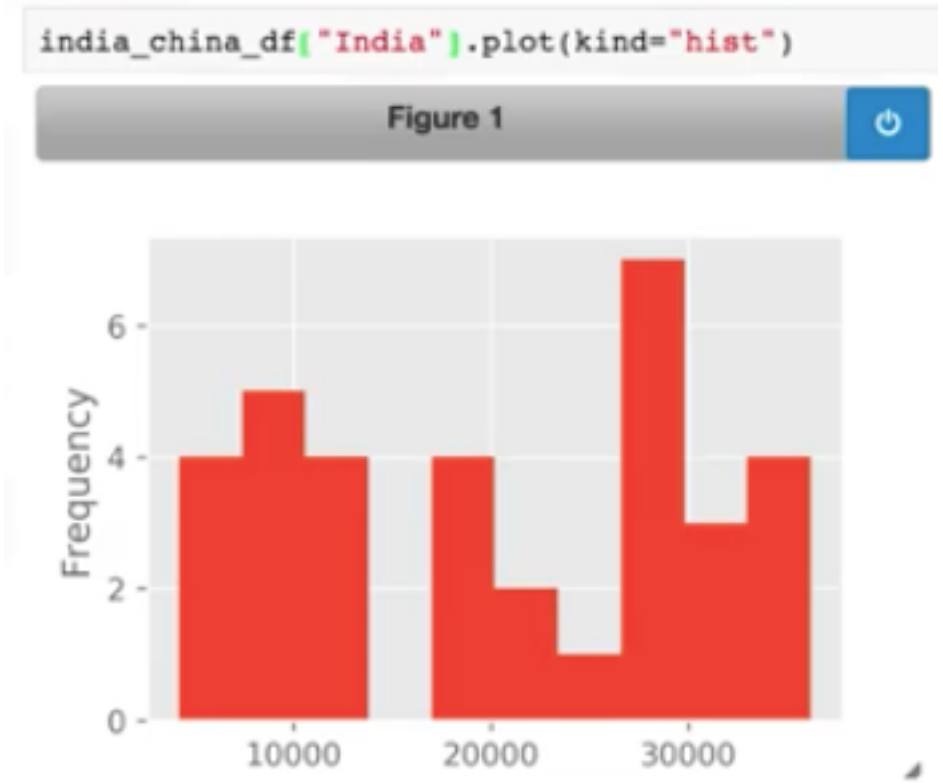
All we have to do is call the plot function on this dataframe which we called India_China_df and set the parameter kind to line

```
india_china_df.plot(kind="line")
```



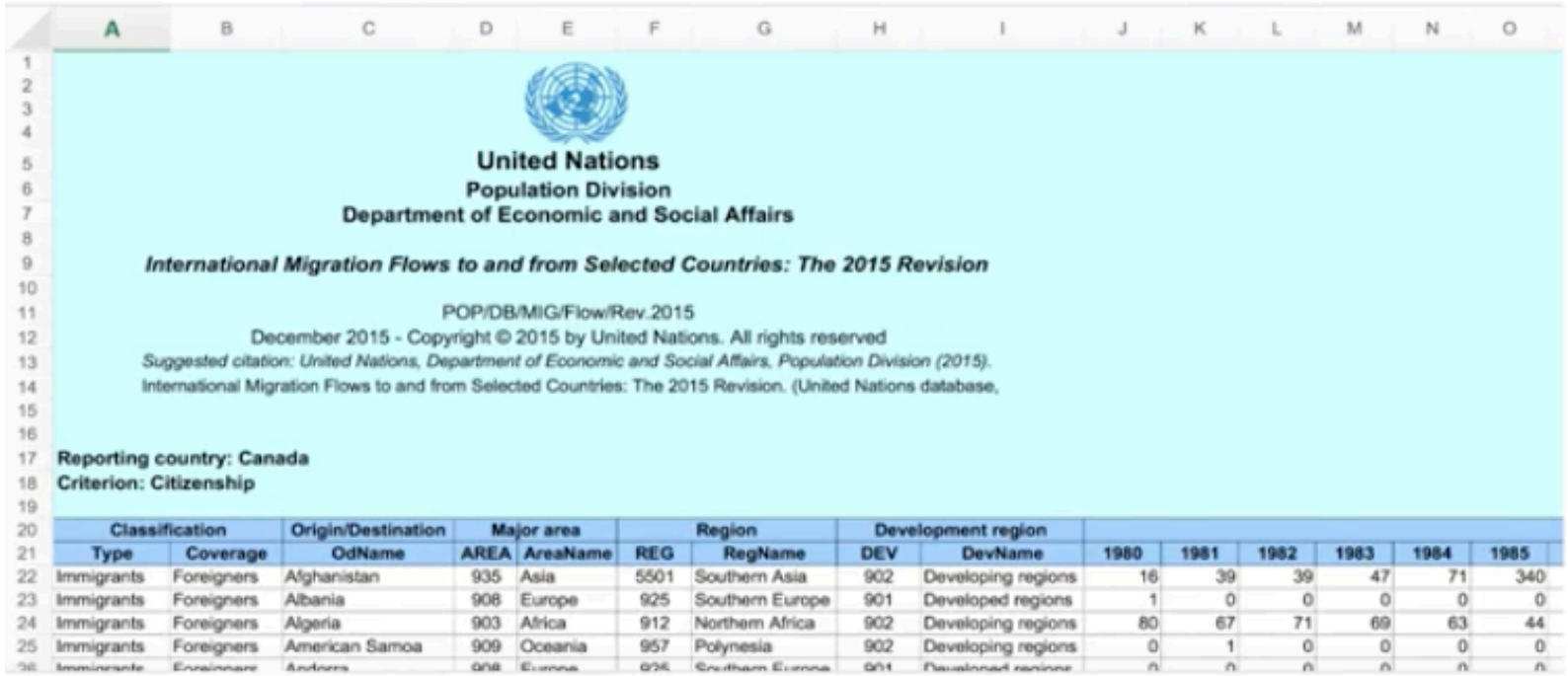
Matplotlib - Pandas

- Plotting a histogram of the data is not any different.
- So say we would like to plot a histogram of the India column in our dataframe. All we have to do is call the plot function on that column and set the parameter kind to hist, for histogram.



Dataset

- We will learn more about the dataset that we will be using for this course.
- The population division of the United Nations compiled immigration data pertaining to 45 countries.
- The data consist of the total number of immigrants from all over the world to each of the 45 countries as well as other metadata pertaining to the immigrants countries of origin.
- In this course, we will focus on immigration to Canada



United Nations
Population Division
Department of Economic and Social Affairs

International Migration Flows to and from Selected Countries: The 2015 Revision

POP/DB/MIG/Flow/Rev.2015

December 2015 - Copyright © 2015 by United Nations. All rights reserved

Suggested citation: United Nations, Department of Economic and Social Affairs, Population Division (2015).

International Migration Flows to and from Selected Countries: The 2015 Revision. (United Nations database,

Reporting country: Canada

Criterion: Citizenship

Classification		Origin/Destination	Major area		Region		Development region						
Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	1981	1982	1983	1984	1985
Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	39	39	47	71	340
Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	0	0	0	0	0
Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	67	71	69	63	44
Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	1	0	0	0	0

Read Dataset into Panda Dataframe

Throughout this course, we will be using pandas for any analysis of the data before creating any visualizations.

We will need to import the Pandas library as well as the xlrd library, which is required to extract data from Excel spreadsheet files.

```
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
from __future__ import print_function # adds compatibility to python 2
```

```
# install xlrd
!pip install xlrd

print('xlrd installed!')
```

Read Dataset into Panda Dataframe

Then we call the pandas function read_excel to read the data into a pandas dataframe and let's name this dataframe df_can.

```
df_can = pd.read_excel(
    'https://ibm.box.com/shared/static/lw190pt9zpy5bd1ptyg2aw15awomz9pu.xlsx',
    sheetname="Canada by Citizenship",
    skiprows=range(20),
    skip_footer = 2)
```

Notice how we're skipping the first twenty rows to read only the data corresponding to each country.

Read Dataset into Panda Dataframe

```
df_can.head()
```

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	...	1450	1223	856	702	560	716	561	539	620	603
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	...	3616	3626	4807	3623	4005	5393	4752	4325	3774	4331
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	...	0	0	1	0	0	0	0	0	0	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	...	0	0	1	1	0	0	0	0	1	1

Basic Visualization

Line plots

Area plots

Histogram

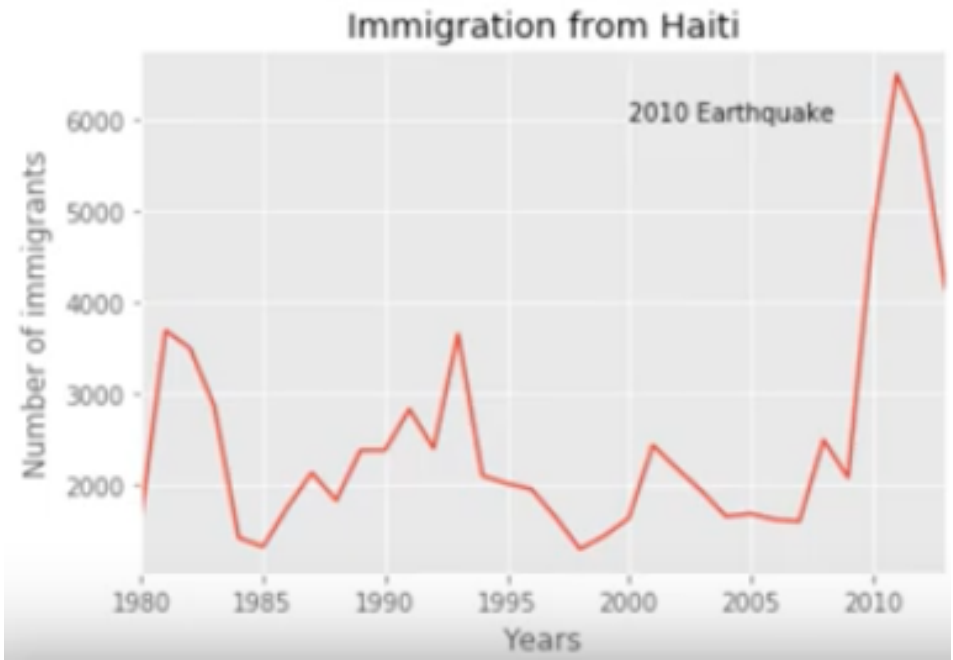
Bar charts

Line Plots

Line plot is a plot in the form of a series of data points connected by straight line segments.

The best use case for a line plot is when you have a continuous dataset and you're interested in visualizing the data over a period of time.

For example, say we're interested in the trend of immigrants from Haiti to Canada.We can generate a line plot and the resulting figure will depict the trend of Haitian immigrants to Canada from 1980 to 2013.



Line Plots

Based on the line plot, we can then research for justifications of obvious anomalies or changes

From previous plot, we see that there is a spike of immigration from Haiti to Canada in 2010.

A quick Google search for major events in Haiti in 2010 would return the tragic earthquake that took place in 2010, and therefore this influx of immigration to Canada was mainly due to that tragic earthquake.

Line Plots

Before we generate line plot, let's process the dataframe so that the country name becomes the index of each row. Also let's add an extra column which represents the cumulative sum of annual immigration from each country from 1980 to 2013. And let's name our dataframe df_canada.

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Country																					
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203	2635	2004	58639
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223	856	702	560	716	561	539	620	603	15699
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626	4807	3623	4005	5393	4752	4325	3774	4331	69439
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	6
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2	...	0	1	1	0	0	0	0	1	1	15

Line Plots in Python

First, we import Matplotlib as mpl and its scripting interface as plt.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Then, we call the plot function on the row corresponding to Haiti and we set kind equals line to generate a line plot.

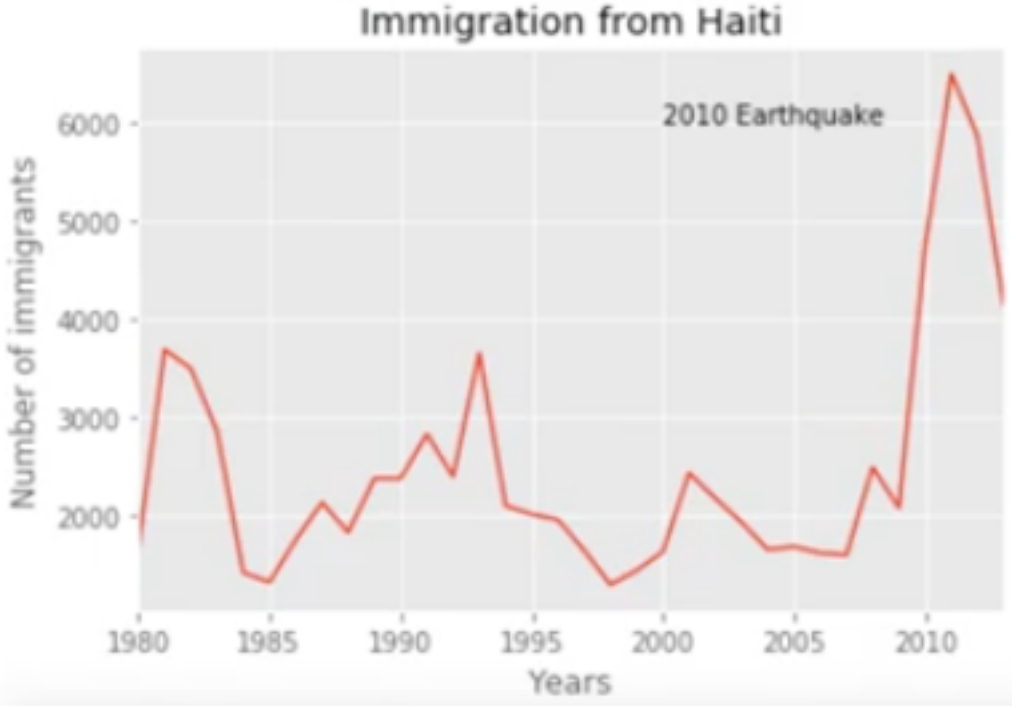
```
years = list(map(str, range(1980, 2014)))

df_canada.loc['Haiti', years].plot(kind = 'line')
plt.title('Immigration from Haiti')
plt.ylabel('Number of immigrants')
plt.xlabel('Years')

plt.show()
```

Note that we used years which is a list containing string format of years from 1980 to 2013 in order to exclude the column of total immigration that we added.

Line Plots in Python

Note that the previous code to generate the line plot using the magic function % matplotlib with the inline backend.

And there you have it: a line plot that depicts immigration from Haiti to Canada from 1980 to 2013.



Practice

Introduction and Line Plots Practice

Area Plots

Area plot also known as an area chart or graph is a type of plot that depicts accumulated totals using numbers or percentages over time.

It is based on the line plot and is commonly used when trying to compare two or more quantities.

Let's try to generate area plots for the countries with the highest number of immigration to Canada.

We can try to find these countries by sorting our dataframe in descending order of cumulative total immigration from 1980 to 2013

We use the sort_values function to sort our dataframe in descending order

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

Area Plots

So it turns out that India followed by China then the UK, Philippines, and Pakistan are the top five countries with the highest number of immigration to Canada.

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Country																					
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36210	33848	28742	28261	29456	34235	27509	30933	33087	691904
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	...	42584	33518	27642	30037	29622	30391	28502	33024	34129	659962
United Kingdom of Great Britain and Northern Ireland	Europe	Northern Europe	Developed regions	22045	24796	20620	10015	10170	9564	9470	...	7258	7140	8216	8979	8876	8724	6204	6195	5827	551500
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...	18139	18400	19837	24887	28573	38617	36765	34315	29544	511391
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314	13127	10124	8994	7217	6811	7468	11227	12603	241600

Area Plots

Before we generate area plot, we need to create a new dataframe of only these five countries and we need to exclude the total column.

More importantly, to generate the area plots for these countries, we need the years to be plotted on the horizontal axis and the annual immigration to be plotted on the vertical axis.

Note that Matplotlib plots the indices of a dataframe on the horizontal axis, and with the dataframe as shown, the countries will be plotted on the horizontal axis

To fix this, we need to take the transpose of the dataframe.

After we sort our dataframe in descending order of cumulative annual immigration, we create a new dataframe of the top five countries and let's call it df_top5

We then select only the columns representing the years 1980 to 2013 in order to exclude the total column before applying the transpose method.

```
df_top5 = df_canada.head()
df_top5 = df_top5[years].transpose()
```

Area Plots

The resulting dataframe is exactly what we want, with five columns where each column represents one of the top five countries and the years being the indices.

Country	India	China	United Kingdom of Great Britain and Northern Ireland	Philippines	Pakistan
1980	8880	5123	22045	6051	978
1981	8670	6682	24796	5921	972
1982	8147	3308	20620	5249	1201
1983	7338	1863	10015	4562	900
1984	5704	1527	10170	3801	668

Area Plots

Now we can generate area plot using dataframe df_top5

First, we import Matplotlib as mpl and its scripting interface as plt.

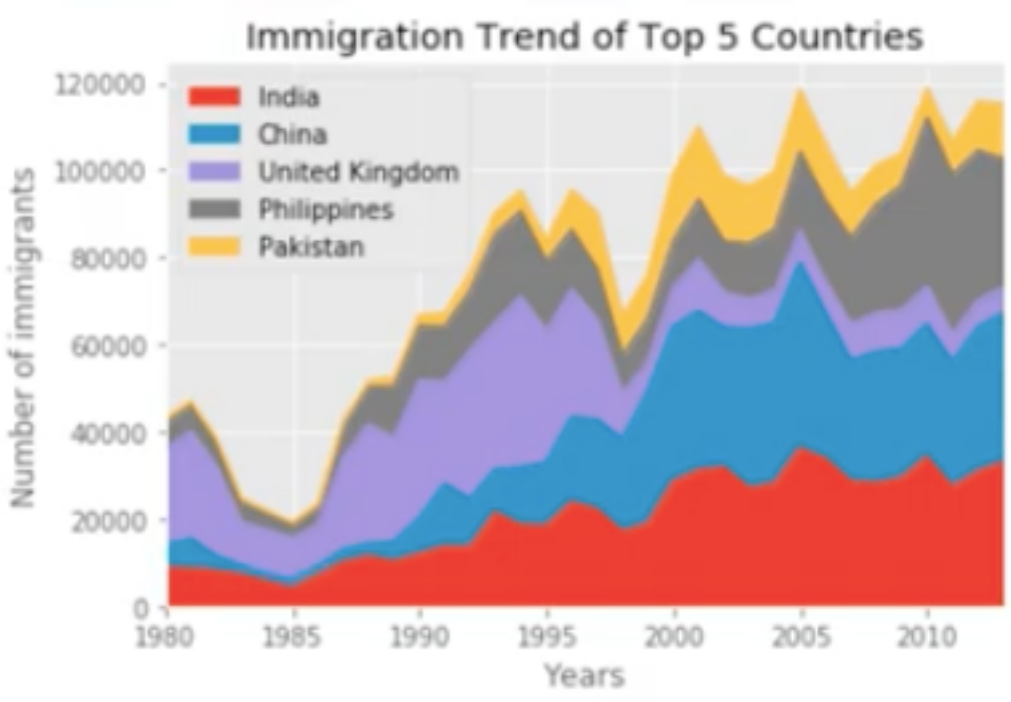
```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Then we call the plot function on the dataframe df_top5 and we set kind equals area to generate an area plot. Note that here we're generating the area plot using the inline backend.

```
df_top5.plot(kind='area')

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

plt.show()
```

Area Plots

This is an area plot that depicts the immigration trend of the five countries with the highest immigration to Canada from 1980 to 2013.



Histogram

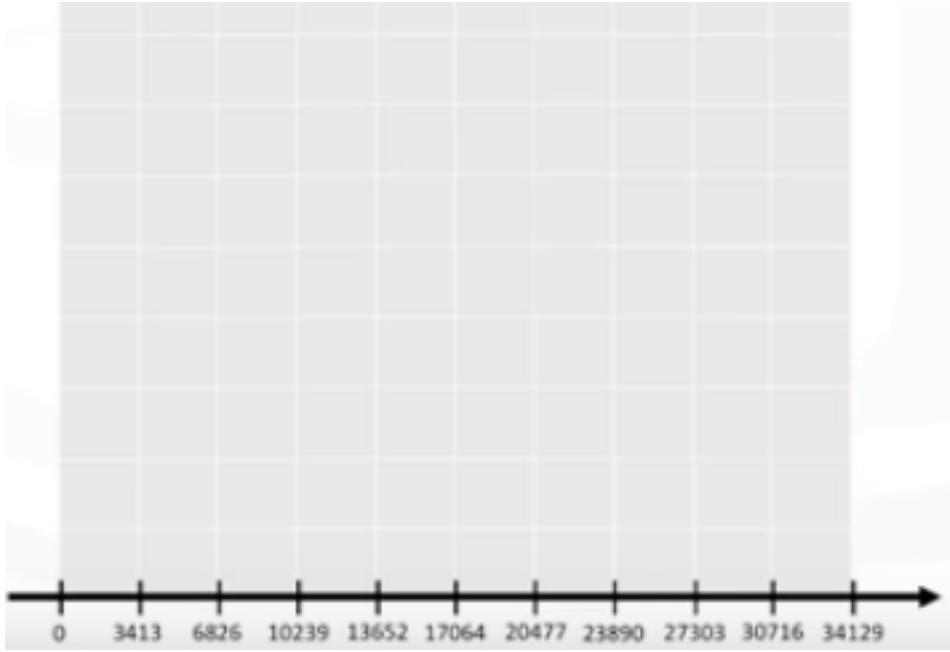
A histogram is a way of representing the frequency distribution of a numeric dataset.

The way it works is it partitions the spread of the numeric data into bins, assigns each datapoint in the dataset to a bin, and then counts the number of datapoints that have been assigned to each bin.

So the vertical axis is actually the frequency or the number of datapoints in each bin.

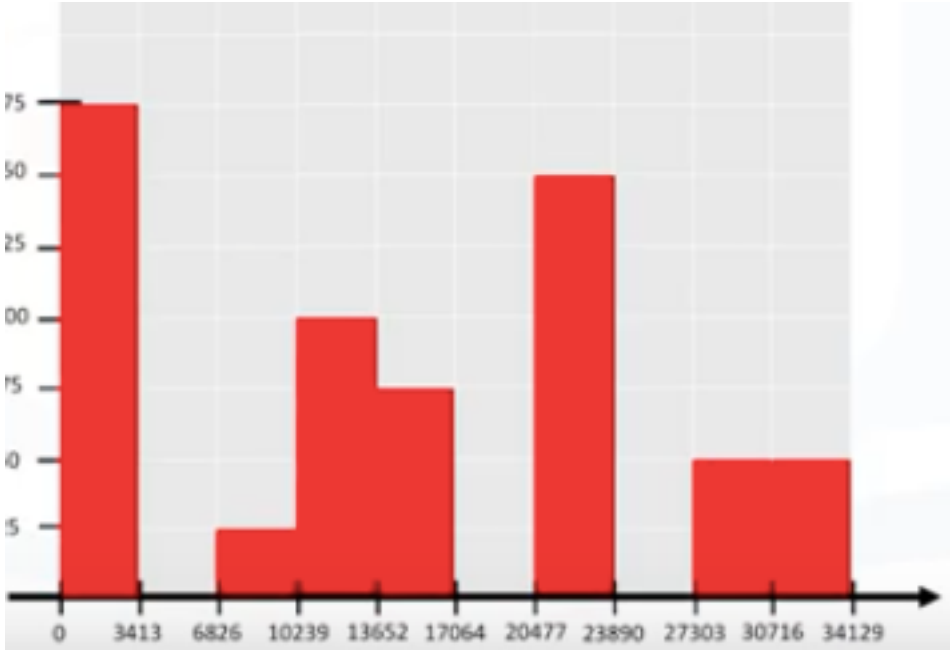
Histogram

- For example, let's say the range of the numeric values in the dataset is 34,129.
- Now, the first step in creating the histogram is partitioning the horizontal axis in, say, ten bins of equal width and then we construct the histogram by counting how many datapoints have a value that is between the limits of the first bin, the second bin, the third bin, and so on.



Histogram

- Say the number of datapoints having a value between 0 and 3,413 is 175, then we draw a bar of that height for this bin.
- We repeat the same thing for all the other bins, and if no datapoints fall into a bin then that bin would have a bar of height 0



Histogram

- We will create a histogram in Matplotlib using dataframe df_canada
- We will visualize the distribution of immigrants to Canada in the year 2013. The simplest way to do that is to generate a histogram of the data in column 2013

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Country																					
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3436	3009	2652	2111	1746	1758	2203	2635	2004	58639
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1223	856	702	560	716	561	539	620	603	15699
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3626	4807	3623	4005	5393	4752	4325	3774	4331	69439
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	6
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2	...	0	1	1	0	0	0	0	1	1	15

Histogram

First, we import Matplotlib as mpl and its scripting interface as plt.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

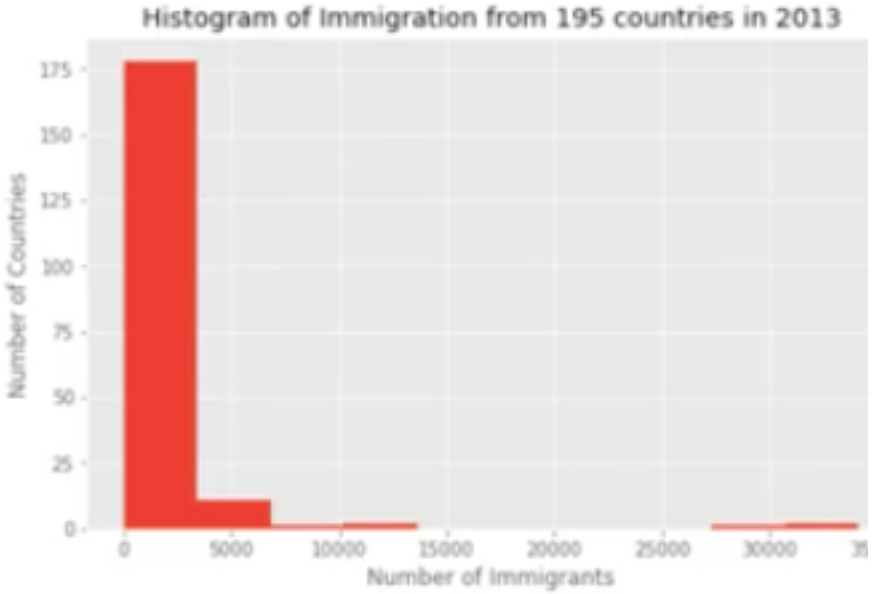
Then we call the plot function on the data in column 2013 and we set kind equals hist to generate a histogram. Then to complete the figure, we give it a title and we label its axes.

Finally, we call the show function to display the figure.

```
count, bin_edges = np.histogram(df_canada['2013'])

df_canada['2013'].plot(kind='hist', xticks = bin_edges)

plt.title('Histogram of Immigration from 195 countries in 2013')
plt.ylabel('Number of Countries')
plt.xlabel('Number of Immigrants')

plt.show()
```

Histogram

A histogram that depicts the distribution of immigration to Canada in 2013 but notice how the bins are not aligned with the tick marks on the horizontal axis. This can make the histogram hard to read.



Histogram

So let's try to fix this in order to make our histogram more effective.

One way to solve this issue is to borrow the histogram function from the Numpy library.

So as usual we start by importing Matplotlib and its scripting interface, but this time we also import the Numpy library.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

Histogram

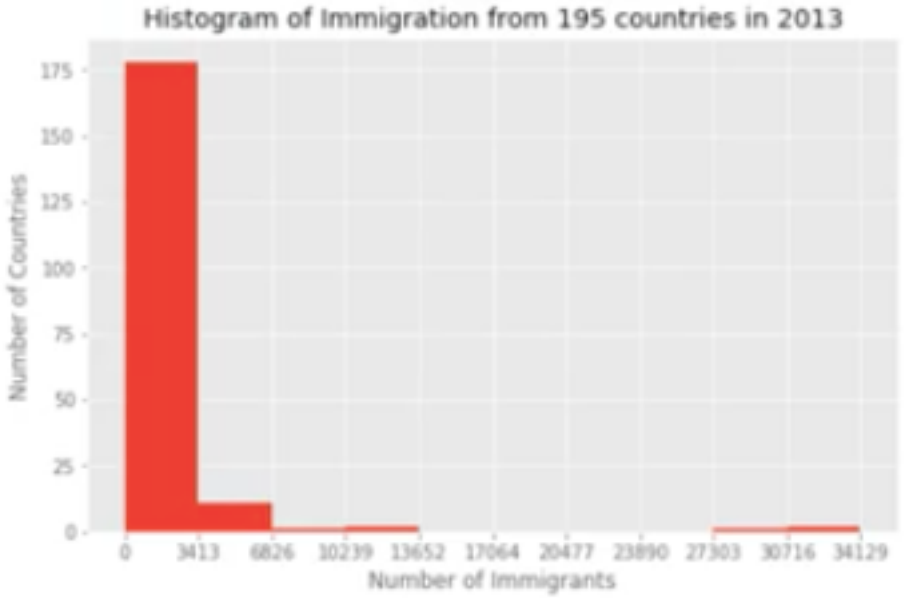
Then we call the Numpy histogram function on the data in column 2013.

What this function is gonna do is it is going to partition the spread of the data in column 2013 into 10 bins of equal width, compute the number of datapoints that fall in each bin, and then return this frequency of each bin which we're calling count here and the bin edges which we're calling bin_edges. We then pass these bin edges as an additional parameter in our plot function to generate the histogram.

```
count, bin_edges = np.histogram(df_canada['2013'])

df_canada['2013'].plot(kind='hist', xticks = bin_edges)

plt.title('Histogram of Immigration from 195 countries in 2013')
plt.ylabel('Number of Countries')
plt.xlabel('Number of Immigrants')

plt.show()
```

Histogram

Here is a better histogram whose bin edges are aligned with the tick marks on the horizontal axis.



Bar Charts

Unlike a histogram, a bar chart commonly used to compare the values of a variable at a given point in time.

For example, say we're interested in visualizing in a discrete fashion how immigration from Iceland to Canada looked like from 1980 to 2013.

One way to do that is by building a bar chart where the height of the bar represents the total immigration from Iceland to Canada in a particular year.

We will generate bar chart in Matplotlib using dataframe df_canada

Bar Charts

As usual, we start by importing Matplotlib and its scripting interface.

Then we use the years variable to create a new dataframe; let's name it df_Iceland, which includes the data pertaining to annual immigration from Iceland to Canada and excluding the total column.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
years = list(map(str, range(1980, 2014)))

df_iceland = df_canada.loc['Iceland', years]
```

Then we call the plot function on df_iceland and we set kind equals bar to generate a bar chart.

Then to complete the figure we give it a title and we label its axes.

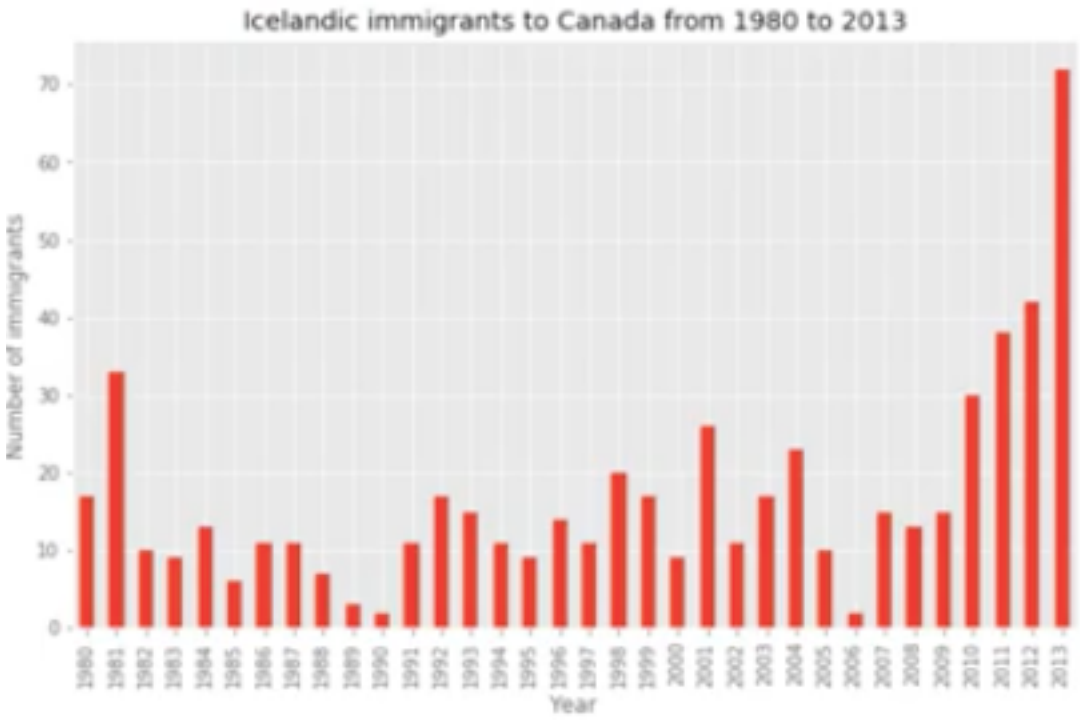
Finally, we call the show function to display the figure.

```
df_iceland.plot(kind='bar')

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

plt.show()
```

Bar Charts

Here is a bar chart that depicts the immigration from Iceland to Canada from 1980 to 2013.

By examining the bar chart we notice that immigration to Canada from Iceland has seen an increasing trend since 2010.



Practice

Basic Visualization Practice

Specialized Visualization Tools

Pie Charts

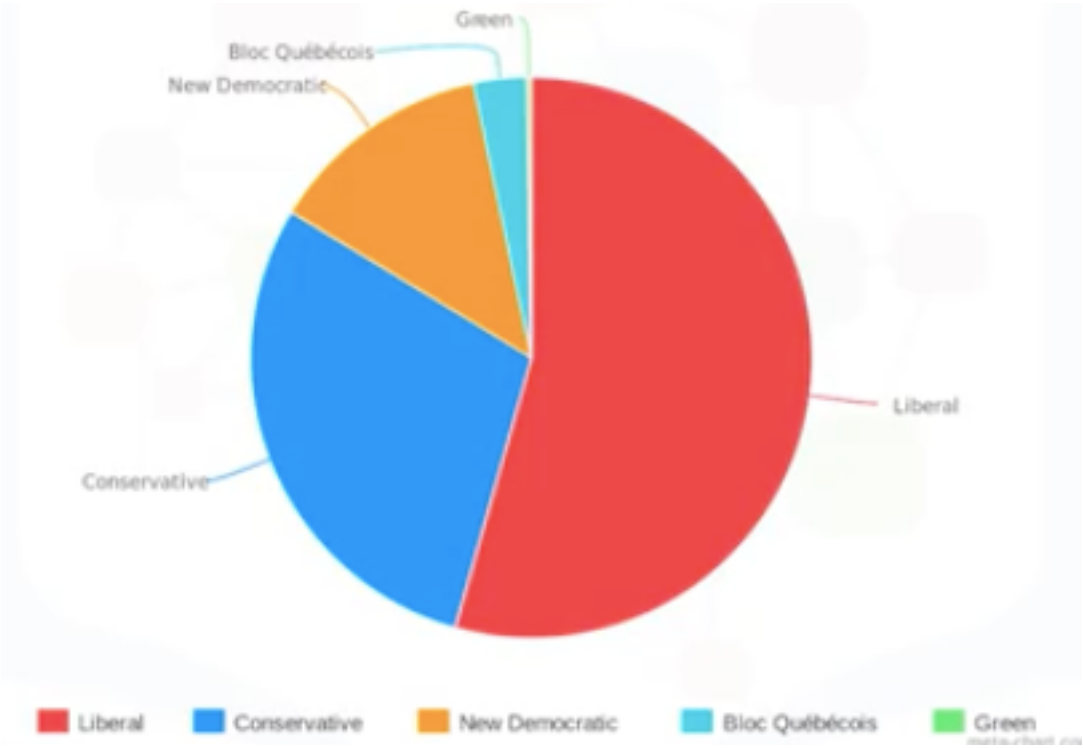
Box Plots

Scatter Plots

Pie Charts

A pie chart is a circular statistical graphic divided into slices to illustrate numerical proportion.

For example, here is a pie chart of the Canadian federal election back in 2015 were the Liberals in red won more than 50% of the seats in the House of Commons. That is why the red color occupies more than half of the circle.



Pie Charts

The first step is to group our data by continent using the continent column and we use pandas for this. We call the pandas groupby function on df_canada and we sum the number of immigrants from the countries that belong to the same continent. Name it as df_continents

```
df_continents = df_canada.groupby('Continent', axis = 0).sum()
```

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Continent																					
Africa	3951	4363	3819	2671	2639	2650	3782	7494	7552	9894	...	27523	29188	28284	29890	34534	40892	35441	38083	38543	618948
Asia	31025	34314	30214	24696	27274	23850	28739	43203	47454	60256	...	159253	149054	133459	139894	141434	163845	146894	152218	155075	3317794
Europe	39760	44802	42720	24638	22287	20844	24370	46698	54726	60893	...	35955	33053	33495	34692	35078	33425	26778	29177	28691	1410947
Latin America and the Caribbean	13081	15215	16769	15427	13678	15171	21179	28471	21924	25060	...	24747	24676	26011	26547	26867	28818	27856	27173	24950	765148
Northern America	9378	10030	9074	7100	6661	6543	7074	7705	6469	6790	...	8394	9613	9463	10190	8995	8142	7677	7892	8503	241142
Oceania	1942	1839	1675	1018	878	920	904	1200	1181	1539	...	1585	1473	1693	1834	1860	1834	1548	1679	1775	55174

Pie Charts

We start with the usual, importing Matplotlib as mpl and its scripting layer the pyplot interface as plt.

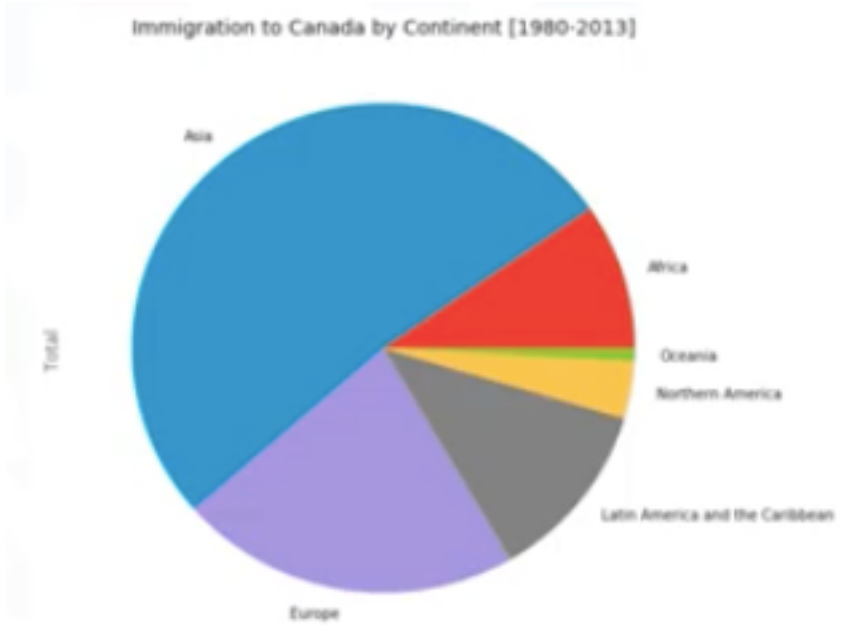
Then we call the plot function on column total of the dataframe df_continents and we set kind equals pie to generate a pie chart. Then to complete the figure we give it a title. Finally we call the show function to display the figure.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
df_continents['Total'].plot(kind='pie')

plt.title('Immigration to Canada by Continent [1980-2013]')

plt.show()
```

Pie Charts

Here is a pie chart that depicts each continent’s proportion of immigration to Canada from 1980 to 2013.



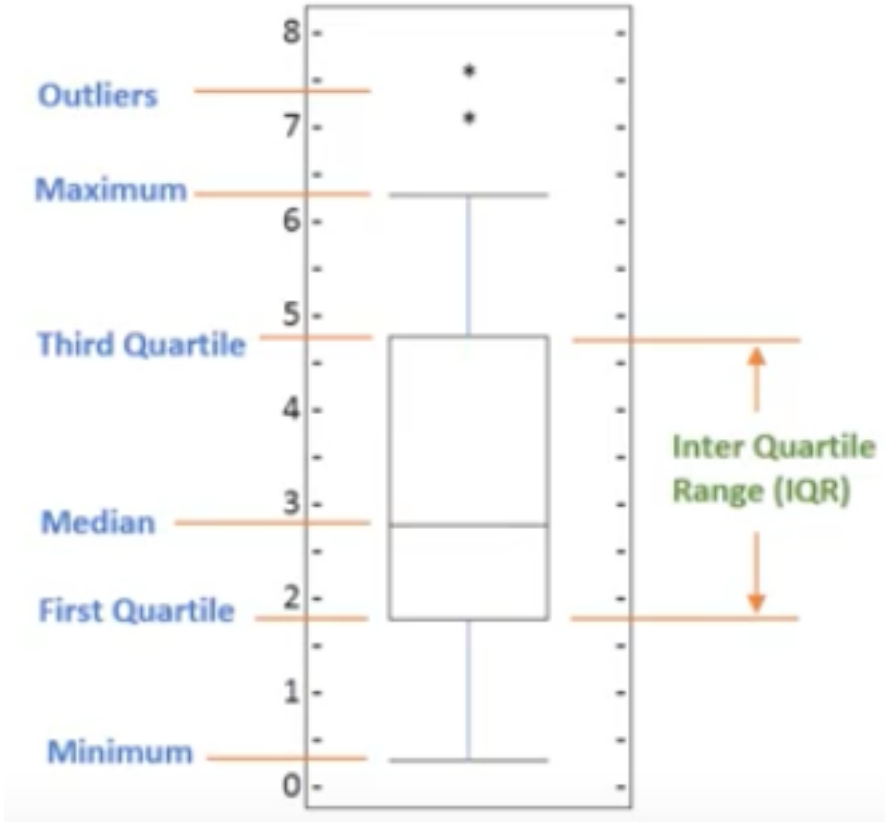
Pie Charts

There are some very vocal opponents to the use of pie charts under any circumstances. Most argue that pie charts fail to accurately display data with any consistency.

Bar charts are much better when it comes to representing the data in a consistent way and getting the message across.

Box Plots

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



Box Plots

Here are five main dimensions of box plots :

-Minimum : smallest number in the sorted data.
-First Quartile : the point 25% of the way through the sorted data
-Median : median of the sorted data
-Third Quartile : the point 75% of the way through the sorted data
-Maximum : highest number in the sorted data

Box Plots

We will create a box plot to visualize immigration from Japan to Canada.

We start by importing Matplotlib as mpl and the pyplot interface as plt.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Then we create a new dataframe of the data pertaining to Japan and we're excluding the column total using the years variable. Then we transpose the resulting dataframe to make it in the correct format to create the box plot. Let's name this new dataframe df_japan.

```
df_japan = df_canada.loc[['Japan'], years].transpose()
```

Box Plots

We call the plot function on df_japan and we set kind equals box to generate a box plot.

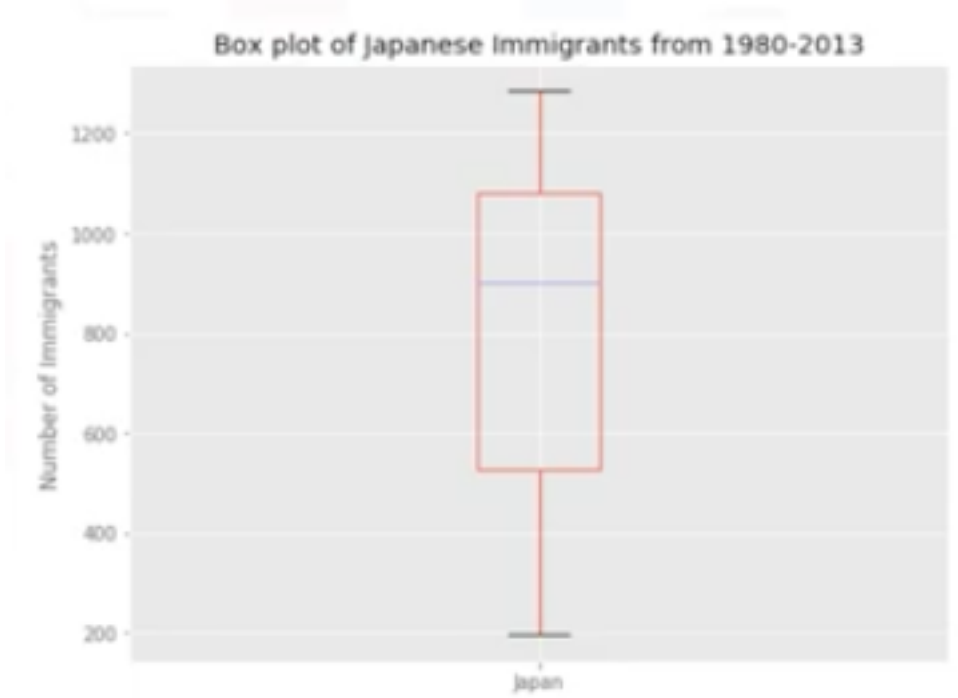
Then to complete the figure we give it a title and we label the vertical axis.

Finally we call the show function to display the figure.

```
df_japan.plot(kind='box')

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

plt.show()
```

Box Plots

Here is a box plot that provides a pleasing distribution of Japanese immigration to Canada from 1980 to 2013.



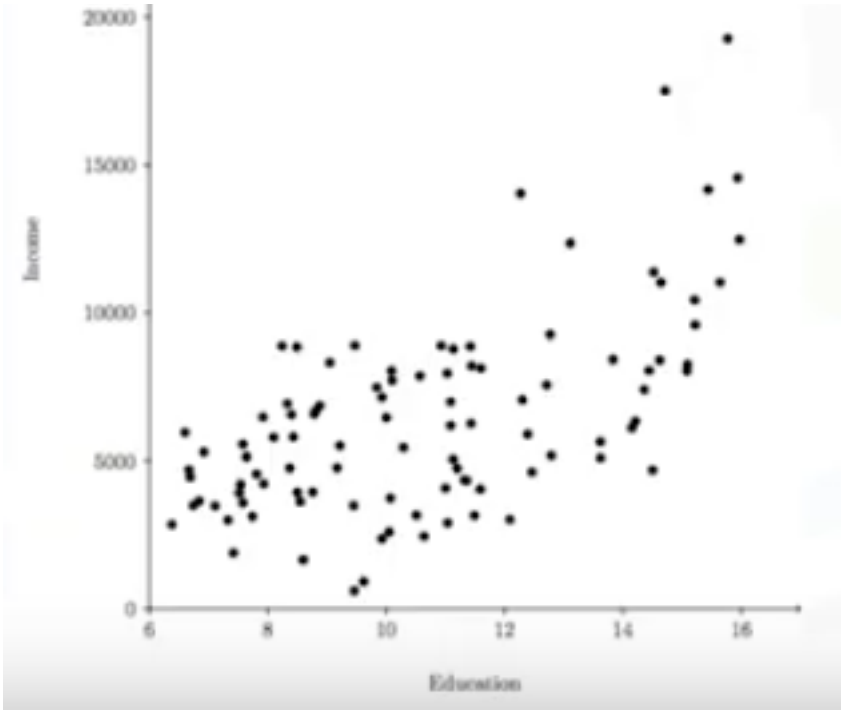
Scatter Plots

A scatter plot is a type of plot that displays values pertaining to typically two variables against each other.

Usually it is a dependent variable to be plotted against an independent variable in order to determine if any correlation between the two variables exists.

Scatter Plots

For example, here is a scatter plot of income versus education and by looking at the plotted data one can conclude that an individual with more years of education is likely to earn a higher income than an individual with fewer years of education.



Scatter Plots

We're interested in plotting a scatter plot of the total annual immigration to Canada from 1980 to 2013.

We first need to create a new dataframe that shows each year and the corresponding total number of immigration from all the countries worldwide as shown here. Name it as df_total

In the lab session, we will walk together through the process of creating df_total from df_Canada

df_total

year	total
1980	99137
1981	110563
1982	104271
1983	75550
1984	73417
.	.
.	.

Scatter Plots

We import Matplotlib as mpl and its scripting layer, the pyplot interface, as plt.

Then we call the plot function on the data frame df_total and we set kind equals scatter to generate a scatter plot.

Unlike the other data visualization tools were only passing the kind parameter was enough to generate the plot, with scatter plots we also need to pass the variable to be plotted on the horizontal axis as the x-parameter and the variable to be plotted on the vertical axis as the y-parameter. In this case, we're passing column year as the x-parameter and column total as the y-parameter.

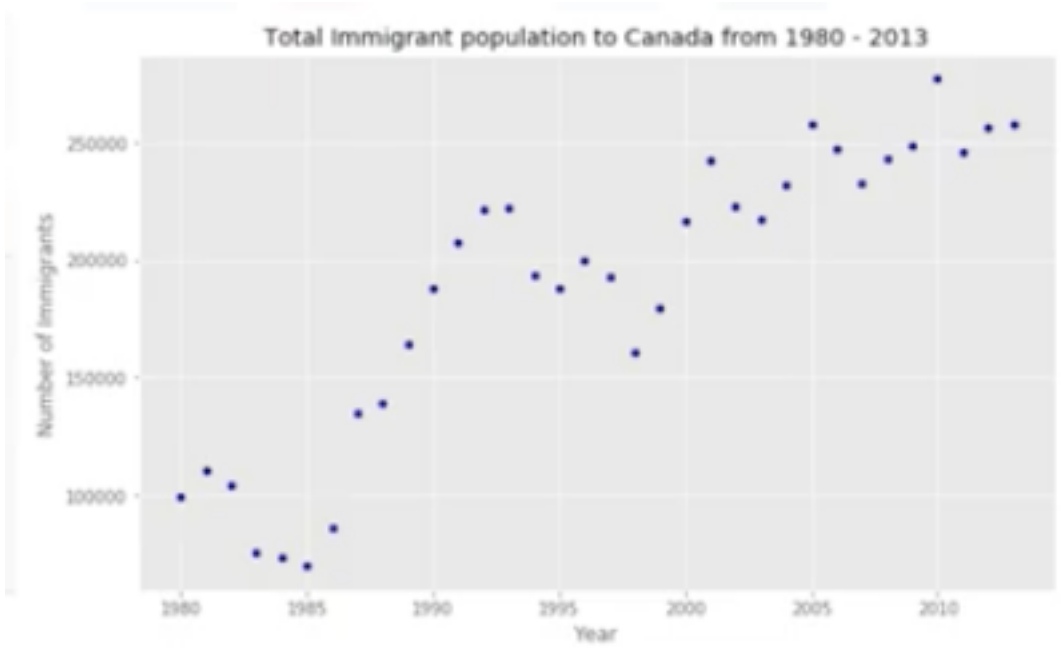
```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
df_total.plot(
    kind='scatter',
    x='year',
    y='total',
)

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

plt.show()
```

Scatter Plots

Here is a scatter plot that shows total immigration to Canada from countries all over the world from 1980 to 2013.

The scatter plot clearly depicts an overall rising trend of immigration with time



Practice

Specialized Visualization Tools Practice

Advanced Visualization Tools

Waffle Charts

Word Clouds

Seaborn and Regression Plots

Waffle Charts

A waffle chart is a great way to visualize data in relation to a whole or to highlight progress against a given threshold.

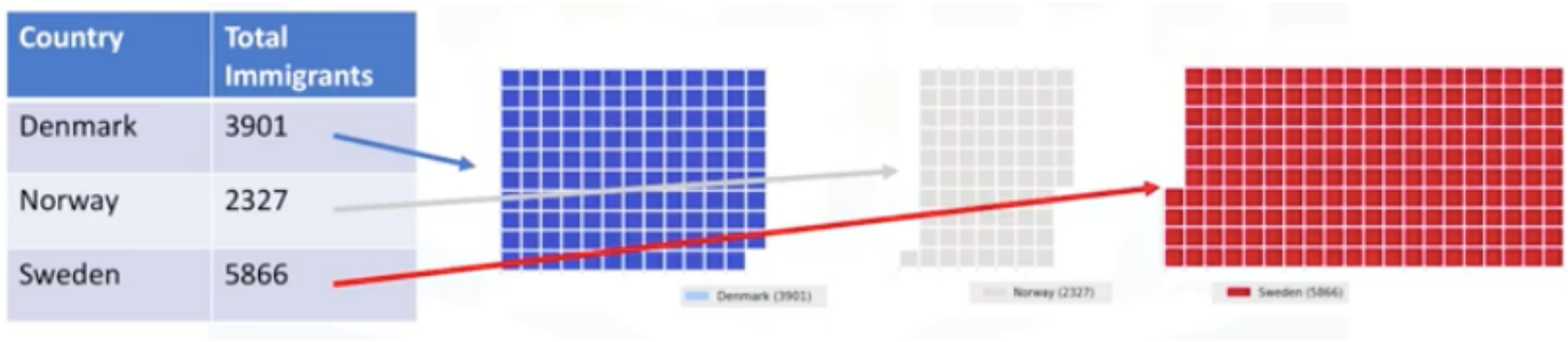
For example, say immigration from Scandinavia to Canada is comprised only of immigration from Denmark, Norway, and Sweden, and we're interested in visualizing the contribution of each of these countries to the Scandinavian immigration to Canada.

Country	Total Immigrants
Denmark	3901
Norway	2327
Sweden	5866

Waffle Charts

The main idea here is for a given waffle chart whose desired height and width are defined; the contribution of each country is transformed into a number of tiles that is proportional to the country's contribution to the total, so that more the contribution the more the tiles, resulting in what resembles a waffle when combined. Hence the name waffle chart.

Country	Total Immigrants
Denmark	3901
Norway	2327
Sweden	5866



Waffle Charts

Unfortunately Matplotlib does not have a built-in function to create waffle charts.

Therefore, in the lab session, we'll walk you through the process of creating your own Python function to create a waffle chart

Word Clouds

A word cloud is simply a depiction of the importance of different words in the body of text.

A word cloud works in a simple way; the more a specific word appears in a source of textual data the bigger and bolder it appears in the world cloud.

Assuming that we didn't know anything about the content of these documents, a word cloud can be very useful to assign a topic to some unknown textual data.

Word Clouds

So given some text data on recruitment, for example, we generate a cloud of words like this.

This cloud is telling us that words such as recruitment, talent, candidates, and so on, are the words that really stand out in these text documents.

Word Clouds

Unfortunately, just like waffle charts, Matplotlib does not have a built-in function to generate word clouds

However, luckily a Python library for cloud word generation that was created by Andreas Mueller is publicly available.

In the lab session we will learn how to use Mueller's word cloud generator and we will also create interesting word clouds superimposed on different background images.

Seaborn

Seaborn is a Python visualization library based on Matplotlib

It was built primarily to provide a high-level interface for drawing attractive statistical graphics, such as regression plots, box plots, and so on.

Seaborn makes creating plots very efficient. Therefore with Seaborn you can generate plots with code that is 5 times less than with Matplotlib

Seaborn

Let's say we have a dataframe called df_total of total immigration to Canada from 1980 to 2013 with the year in one column and the corresponding total immigration in another column

We're interested in creating a scatter plot along with a regression line to highlight any trends in the data.

df_total

year	total
1980	99137
1981	110563
1982	104271
1983	75550
1984	73417
.	.
.	.

Regression Plots with Seaborn

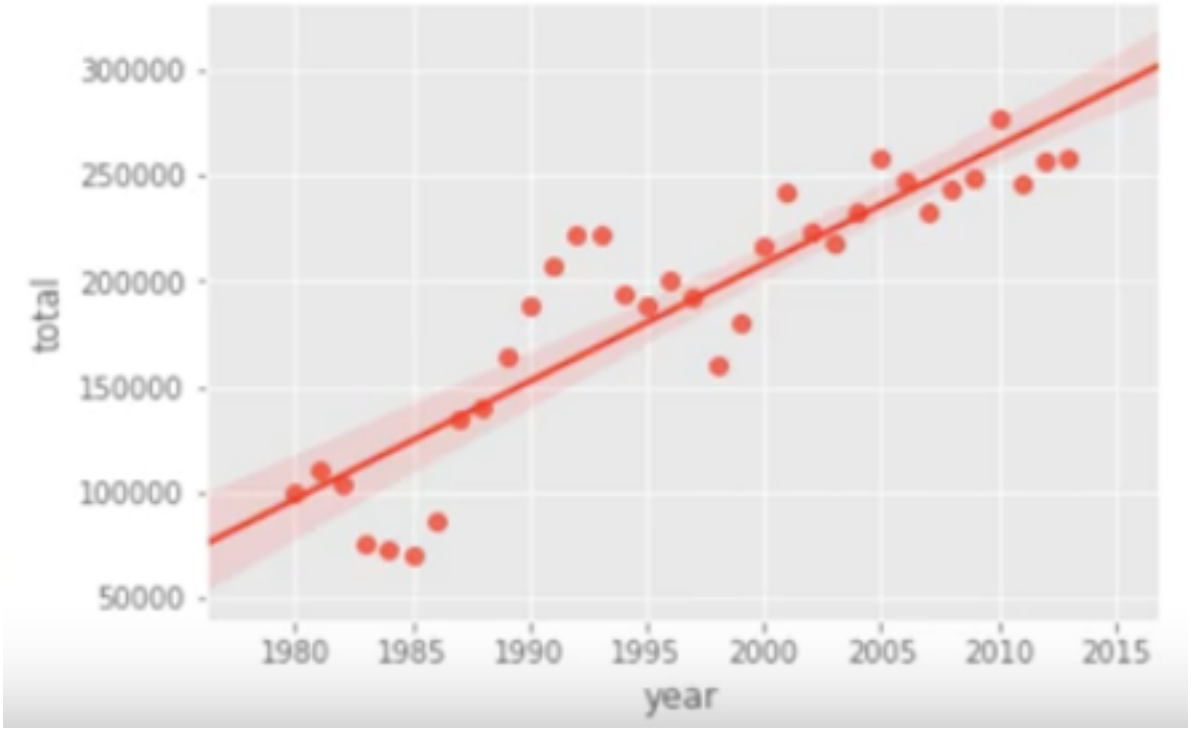
First, import Seaborn and let's import it as sns. Then, we call the Seaborn regplot function.

We basically tell it to use the dataframe df_total and to plot the column year on the horizontal axis and the column total on the vertical axis.

```
import seaborn as sns
ax = sns.regplot(x='year', y='total', data=df_tot)
```

Regression Plots with Seaborn

The output of the previous code is a scatter plot with a regression line and not just that, but also 95% confidence interval.



Regression Plots with Seaborn

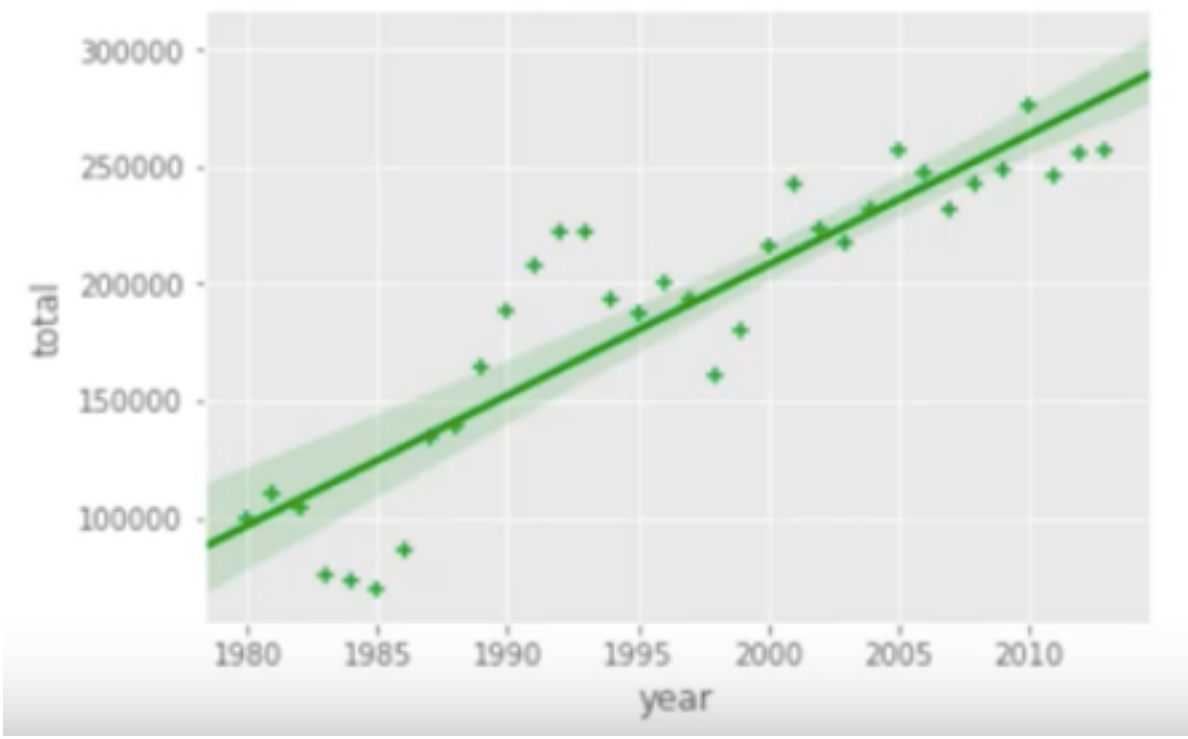
Seaborn's regplot function also accepts additional parameters for any personal customization.

So you can change the color for example using the color parameter. Also, you can change the marker shape as well using the marker parameter.

```
import seaborn as sns
ax = sns.regplot(x='year', y='total', data=df_tot,
                 color='green', marker='+')
```

Regression Plots with Seaborn

We change the color to green and markers to +



Practice

Advanced Visualization Practice

Creating Map and Visualizing Geospatial Data

Folium

Choropleth Maps

Folium

Folium is a powerful data visualization library in Python that was built primarily to help people visualize geospatial data.

With Folium, we can create a map of any location in the world as long as you know its latitude and longitude values.

We can also create maps of different styles such as street level map, stamen map, etc.

Creating World Map with Folium

Creating a world map with Folium is straightforward. You simply call the map function and that is all.

```
# define the world map
world_map = folium.Map()

# display world map
world_map
```

Creating a Map of Canada

To create map of Canada, we pass in the latitude and the longitude values of Canada using the location parameter and with Folium you can set the initial zoom level using the zoom start parameter.

We can change the zoom level after the map is rendered by zooming in or zooming out.

```
# define the world map centered around
# Canada with a low zoom level
world_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4
)

# display world map
world_map
```

Creating a Map – Stamen Toner

Another amazing feature of Folium is that you can create different map styles using the tiles parameter. Let's create a stamen toner map of Canada.

This style is great for visualizing and exploring river meanders and coastal zones.

```
# create a Stamen Toner map of
# the world centered around Canada
world_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4,
    tiles='Stamen Toner'
)

# display map
world_map
```

Creating a Map – Stamen Terrain

Another style is stamen terrain. Let's create a map of Canada in stamen terrain.

This style is great for visualizing hill shading and natural vegetation colors

```
# create a Stamen Toner map of
# the world centered around Canada
world_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4,
    tiles='Stamen Terrain'
)

# display map
world_map
```

Creating a Map – Add Marker

Previously, we learned how to create a world map centred around Canada, so let's create this map again and name it canada_map this time.

```
# generate map of Canada
canada_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4
)
```

Ontario is a Canadian province and contains about 40 percent of the Canadian population. It is considered Canada's most populous province. Let's see how we can add a circular mark to the centre of Ontario.

Creating a Map – Add Marker

To add a marker, we need to create a feature group.

```
# create a feature group
ontario = folium.map.FeatureGroup()
```

Now when a feature group is created, it is empty and that means what's next is to start creating what is called children and adding them to the feature group.

Let's create a child in the form of a red circular mark located at the centre of the Ontario province. We specify the location of the child by passing in its latitude and longitude values.

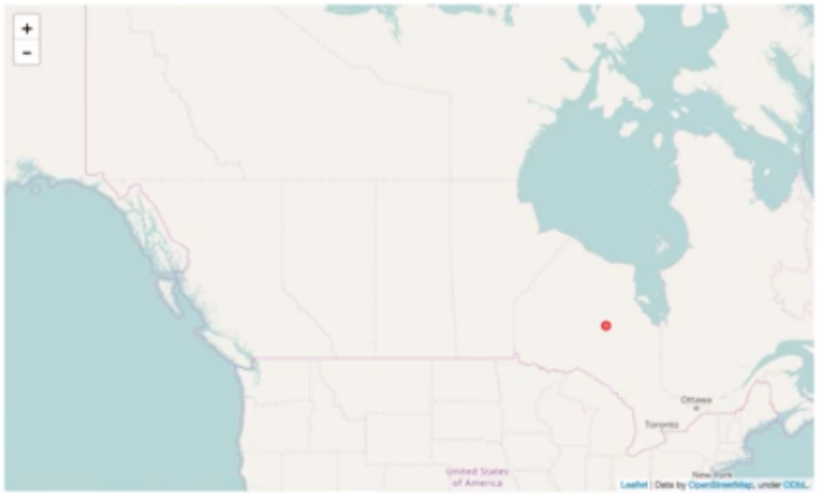
```
# style the feature group
ontario.add_child(
    folium.features.CircleMarker(
    [51.25, -85.32], radius = 5,
    color = "red", fill_color = "Red"
    )
)
```

Creating a Map – Add Marker

Once we're done adding children to the feature group, we add the featured group to the map.

```
# add the feature group to the map
canada_map.add_child(ontario)
```

Here is a red circular mark superimposed on top of the map and added to the centre of the province of Ontario.



Creating a Map – Label Marker

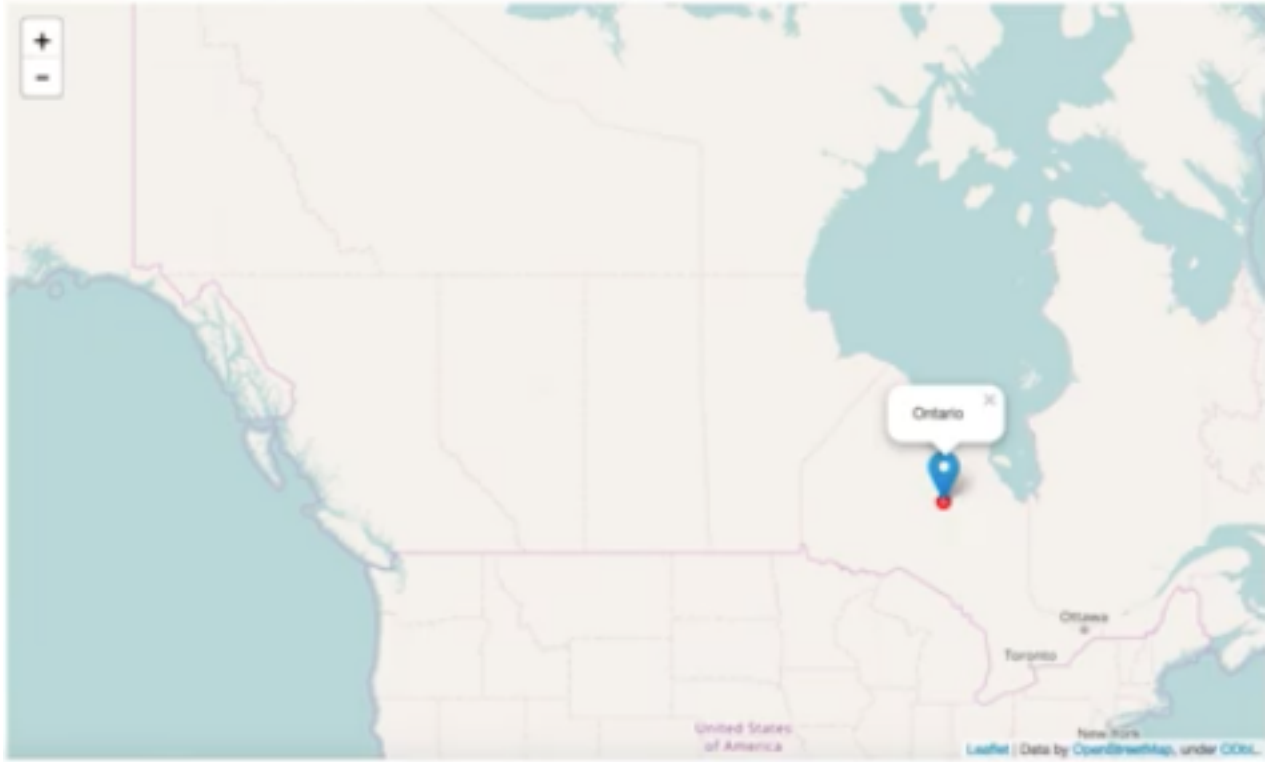
We would label previous marker in order to let other people know what it represents.

To do that, we use the marker function and the pop-up parameter to pass in whatever text we want to add to this marker.

```
# label the marker
folium.Marker([51.25, -85.32],
    popup='Ontario').add_to(canada_map)
```

Creating a Map – Label Marker

Now our marker displays Ontario when clicked on :



Choropleth Maps

A choropleth map is a thematic map in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map, such as population density or per capita income. The higher the measurement the darker the color.

GeoJSON File

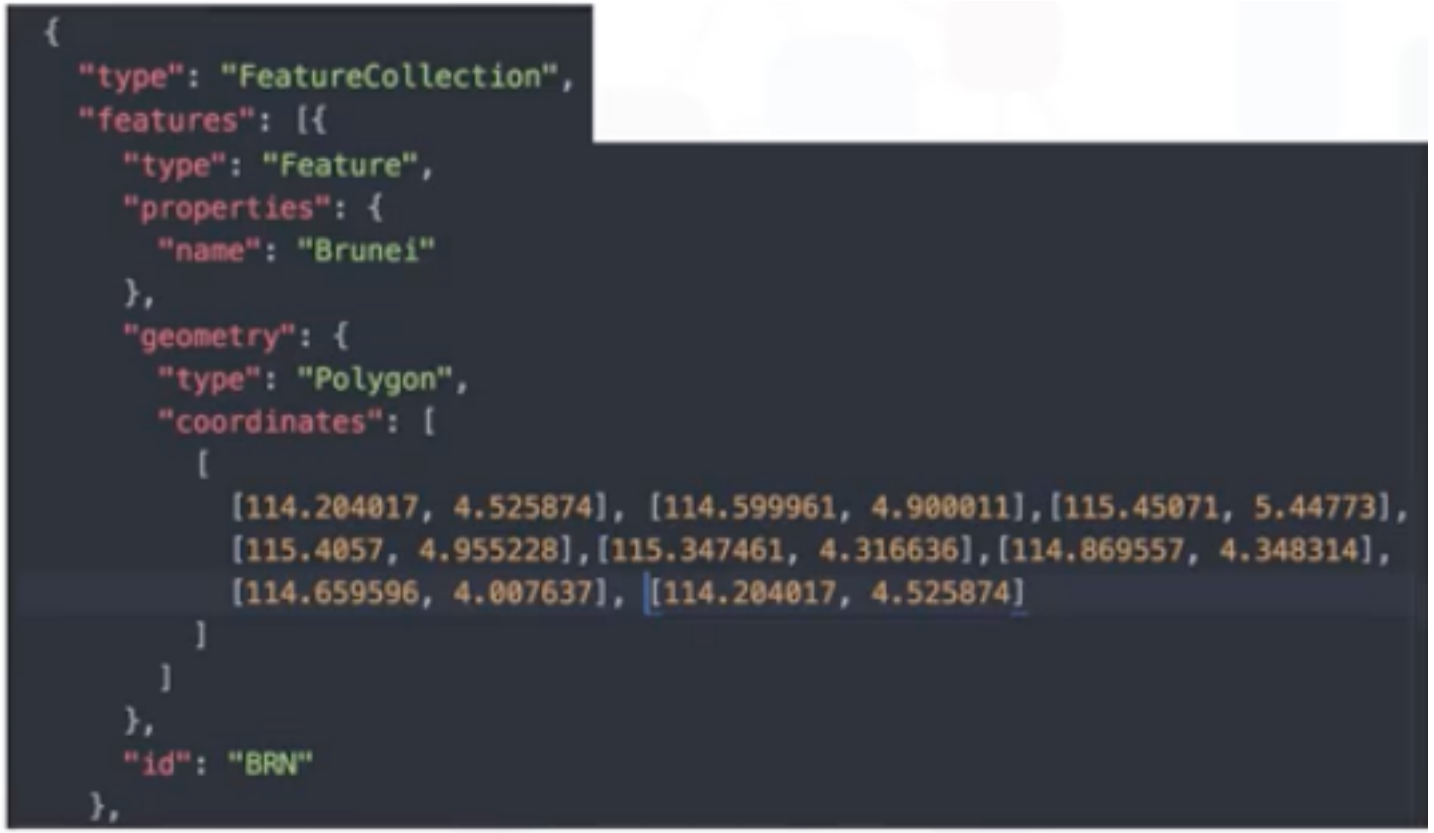
To create a choropleth map of a region of interest, Folium requires a Geo JSON file that includes geospatial data of the region

For a choropleth map of the world, we would need a Geo JSON file that lists each country along with any geospatial data to define its borders and boundaries.

GeoJSON File

Here is an example of what the Geo JSON file would include about each country.

The file includes the country's name, it's ID, geometry shape, and the coordinates that define the country's borders and boundaries.



Create a Choropleth Map

We will generate a choropleth map of the world showing immigration to Canada using dataframe df_Canada

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	...	1450	1223	856	702	560	716	561	539	620	603
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	...	3616	3626	4807	3623	4005	5393	4752	4325	3774	4331
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	...	0	0	1	0	0	0	0	0	0	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	...	0	0	1	1	0	0	0	0	1	1

Create a Choropleth Map

We will create a world map, but this time let's use the mapbox bright tiles set. The result is a world map displaying the name of every country.

```
# create a plain world map
world_map = folium.Map(
    zoom_start=2,
    tiles='Mapbox Bright'
)
```

Create a Choropleth Map

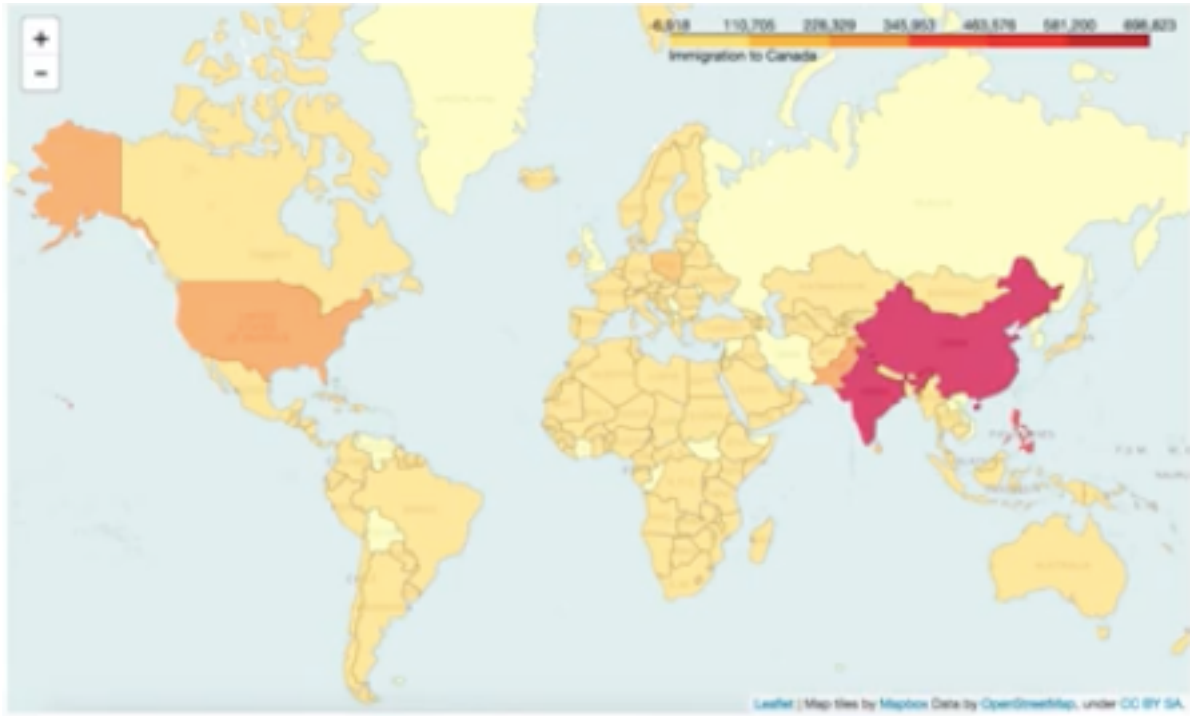
To convert this map into a choropleth map, we first define a variable that points to our Geo JSON file.

Then we apply the choropleth function to our world map, and we tell it to use the columns "Country" and "Total" in our df_canada dataframe, and to use the country names to look up the geospatial information about each country in the Geo JSON file.

```
## geojson file
world_geo = r'world_countries.json'

# generate choropleth map using the total
# population of each country to Canada from
# 1980 to 2013
world_map.choropleth(
    geo_path=world_geo,
    data=df_canada,
    columns=['Country', 'Total'],
    key_on='feature.properties.name',
    fill_color='YlOrRd',
    legend_name='Immigration to Canada'
)
```

Create a Choropleth Map

Here is a choropleth map of Canada showing the intensity of immigration from different countries worldwide :



Practice

Generating Maps Practice

Reference

https://cognitiveclass.ai/courses/data-visualization-with-python