



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

Estimated time needed: 30 minutes

Objectives

After completing this lab you will be able to:

- Explore Matplotlib library further
- Create pie charts, box plots, scatter plots and bubble charts

Table of Contents

1. Exploring Datasets with 'pandas'#0
2. Downloading and Prepping Data'#2
3. Visualizing Data using Matplotlib'#4
4. Pie Charts'#6
5. Box Plots'#8
6. Scatter Plots'#10
7. Bubble Plots'#12

Exploring Datasets with pandas and Matplotlib

Toolkits: The course heavily relies on **pandas** and **Numpy** for data wrangling, analysis, and visualization. The primary plotting library we will explore in the course is **Matplotlib**.

Dataset: Immigration to Canada from 1980 to 2013 - [International migration flows to and from selected countries - The 2015 revision](#) from United Nation's website.

The dataset contains annual data on the flows of international migrants as recorded by the countries of destination. The data presents both inflows and outflows according to the place of birth, citizenship or place of previous / next residence both for foreigners and nationals. In this lab, we will focus on the Canadian Immigration data.

Downloading and Prepping Data

Import primary modules.

```
In [2]: import numpy as np # useful for many scientific computing in Python
import pandas as pd # primary data structure library
```

Let's download and import our primary Canadian Immigration dataset using `pandas`' `read_excel()` method. Normally, before we can do that, we would need to download a module which `pandas` requires reading in Excel files. This module was **openpyxl** (formerly **xlrd**). For our 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.

```
In [3]: df_can = pd.read_excel(
'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Canada_Immigration/Citizenship.xlsx',
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.

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

```
Out [4]:
```

	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		
1	Immigrants	Foreigners	Albania	908	Europe	925	Northern Africa	901	Developed regions	1	...	1450	1223	856	702	560	716	561	539		
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	...	3616	3626	4807	3623	4005	5393	4752	4325		
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	...	0	0	1	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		

5 rows × 43 columns

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

```
In [5]: # print the dimensions of the dataframe
print(df_can.shape)
```

Clear 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 and Area Plots, Histograms, and Bar Plots](#) for a detailed description of this preprocessing.

```
In [6]: # clean up the dataset to remove unnecessary columns (eg. REG)
df_can.drop(['AREA', 'REG', 'DEV', 'Type', 'Coverage'], axis=1, inplace=True)

# let's rename the columns so that they make sense
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'RegName':'Region'}, inplace=True)

# for sake of consistency, let's also make all column labels of type string
df_can.columns = list(map(str, df_can.columns))

# set the country name as index - useful for quickly looking up countries using .loc method
df_can.set_index('Country', inplace=True)

# add total column
df_can['Total'] = df_can.sum(axis=1)

# years that we'll be using in this lesson - useful for plotting later on
years = list(map(str, range(1980, 2014)))
print('data dimensions:', df_can.shape)
```

data dimensions: (195, 38)

Visualizing Data using Matplotlib

Import Matplotlib.

```
In [7]: %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: 3.3.4

Pie Charts

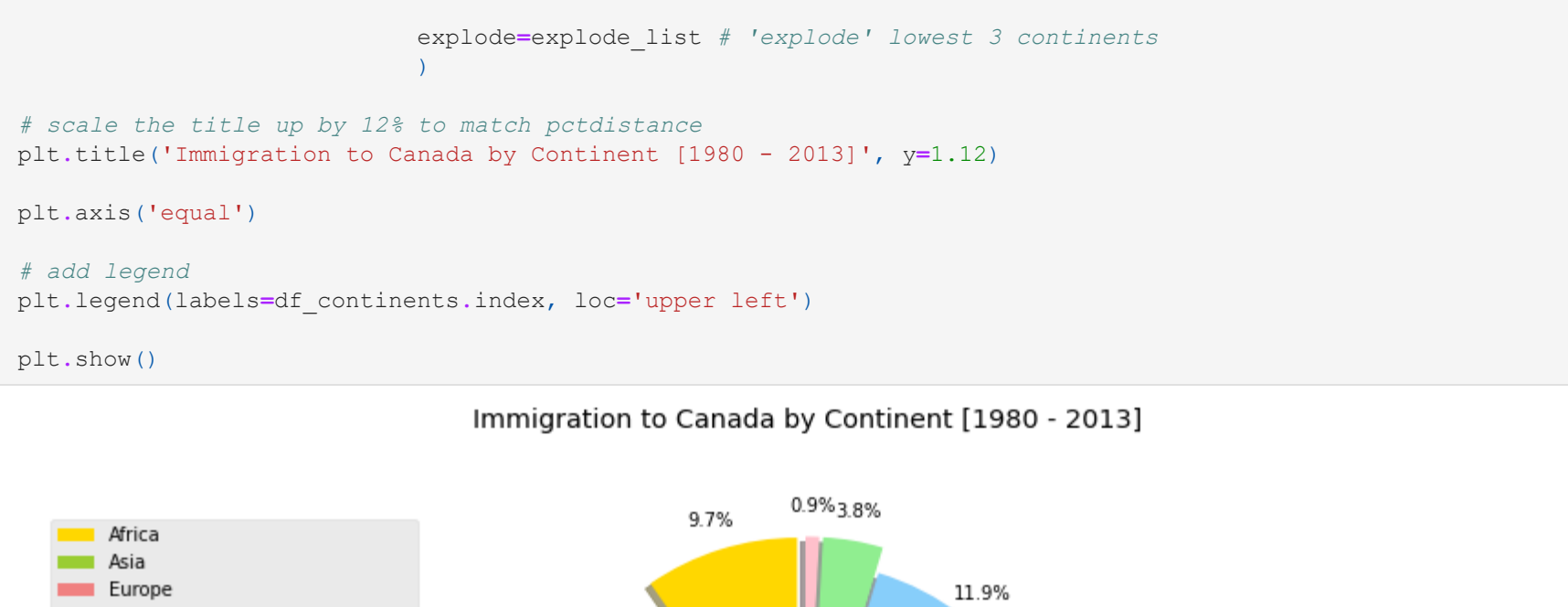
A **pie chart** is a circular graphic that displays numeric proportions by dividing a circle (or pie) into proportional slices. You are most likely already familiar with pie charts as it is widely used in business and media. We can create pie charts in Matplotlib by passing in the `kind=pie` keyword.

Let's use a pie chart to explore the proportion (percentage) of new immigrants grouped by continents for the entire time period from 1980 to 2013.

Step 1: Gather data.

We will use `pandas` `groupby` method to summarize the immigration data by `Continent`. The general process of `groupby` involves the following steps:

1. **Split**: Splitting the data into groups based on some criteria.
2. **Apply**: Applying a function to each group independently. `sum()`, `count()`, `mean()`, `std()` aggregate() apply() etc.
3. **Combine**: Combining the results into a data structure.



```
In [8]: # group countries by continents and apply sum() function
df_continents = df_can.groupby('Continent', axis=0).sum()
```

note: the output of the groupby method is a 'groupby' object.
we can not use it further until we apply a function (eg. sum())
print(type(df_continents['Continent', 'axis=0']))

```
df_continents.head()
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

Continent	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010	2011	2012	2013
Africa	3951	4363	3819	2671	2679	2650	3782	7494	7552	9894	...	27523	29188	28284	29890	34534	40892	35		
Asia	31025	34314	30214	24696	27274	23850	28739	43203	47454	60256	...	159253	149054	133459	139894	141434	163845	146		
Europe	39760	44802	42720	24638	22287	20844	24370	46698	54726	60893	...	35955	33053	33495	34692	35078	33425	26		
Latin America and the Caribbean	13081	15215	16769	15427	13678	15171	21179	28471	21924	25060	...	24747	24676	26011	26547	26867	28818	27		
Northern America	9378	10030	9074	7100	6661	6543	7074	7705	6469	6490	...	8394	9613	9463	10190	8995	8142	7		

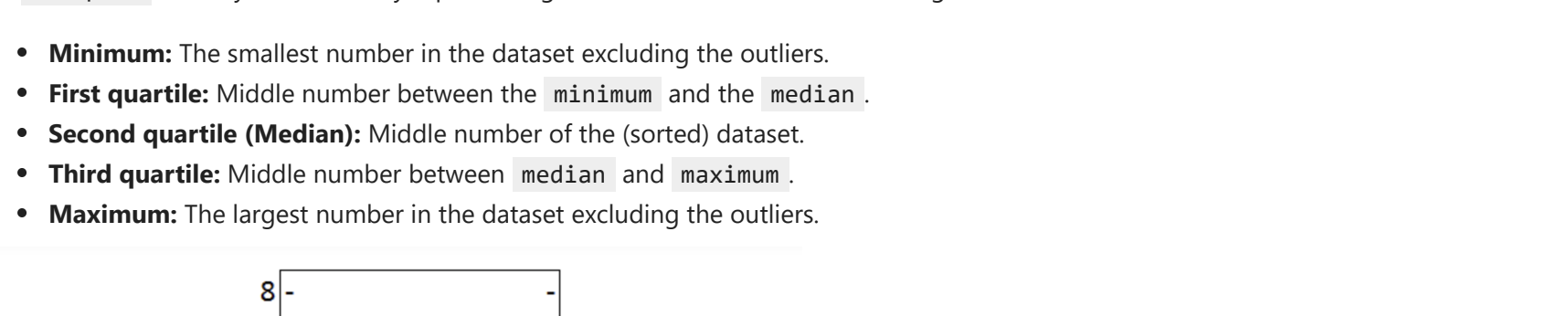
5 rows × 35 columns

Step 2: Plot the data. We will pass in `kind = 'pie'` keyword, along with the following additional parameters:

- `autopct` - is a string or function used to label the wedges with their numeric value. The label will be placed inside the wedge. If it is a formal string, the label will be `float(xpct)`.
- `startangle` - rotates the start of the pie chart by angle degrees counterclockwise from the x-axis.
- `shadow` - Draws a shadow beneath the pie (to give a 3D feel).

```
In [9]: # autopct create % , start angle represent starting point
df_continents['Total'].plot(kind='pie',
figsize=(15, 6),
autopct='%1.1f%%', # add in percentages
startangle=90, # start angle 90° (Africa)
shadow=True, # add shadow
labels=None, # turn off labels on pie chart
pctdistance=1.12, # the ratio between the center of each pie slice and the start
colormap=plt.cm.Paired, # add custom colors
explode=explode_list) # 'explode' lowest 3 continents

plt.title('Immigration to Canada by Continent [1980 - 2013]')
plt.axis('equal') # Sets the pie chart to look like a circle.
plt.show()
```



The above visual is not very clear, the numbers and text overlap in some instances. Let's make a few modifications to improve the visuals:

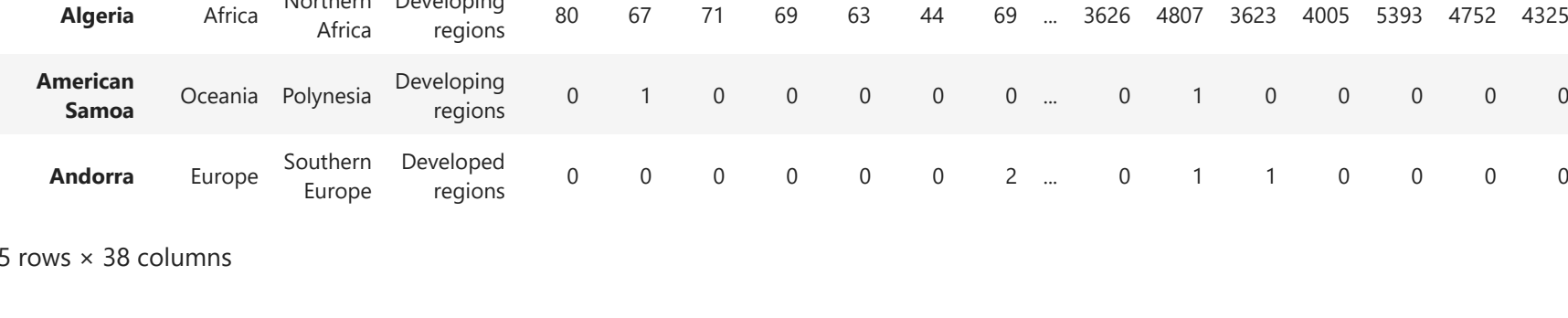
- Remove the text labels on the pie chart by passing in `labels=None` and add it as a separate legend using `plt.legend()`.
- Push out the percentages to sit just outside the pie chart by passing in `pctdistance` parameter.
- Pass in a custom set of colors for continents by passing in `colormap` parameter.
- **Explode** the pie chart to emphasize the lowest three continents (Africa, North America, and Latin America and Caribbean) by passing in `explode=explode_list` parameter.

```
In [10]: colors_list = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink']
explode_list = ['Africa', 'Asia', 'Europe', 'Latin America and the Caribbean', 'Northern America']

df_continents['Total'].plot(kind='pie',
figsize=(15, 6),
autopct='%1.1f%%',
startangle=90,
shadow=True,
labels=None, # turn off labels on pie chart
pctdistance=1.12, # the ratio between the center of each pie slice and the start
colormap=plt.cm.Paired, # add custom colors
explode=explode_list) # 'explode' lowest 3 continents

# scale the title up by 128 to match pctdistance
plt.title('Immigration to Canada by Continent [1980 - 2013]', y=1.12)
plt.axis('equal')

# add legend
plt.legend(labels=df_continents.index, loc='upper left')
plt.show()
```



Question: Using a pie chart, explore the proportion (percentage) of new immigrants grouped by continents in the year 2013.

Note: You might need to play with the `explode` values in order to fix any overlapping slice values.

```
In [11]: ### type your answer here

newdf = df_can.groupby('Continent').sum()
newdf['2013'].plot(kind='pie', figsize=(16,7), autopct='%1.1f%%', startangle=90, shadow=True, colors=colors_list, label=
plt.legend(newdf.index, loc='best')
```

```
Out [11]: %matplotlib.legend.legend at 0x7f979246710>
```

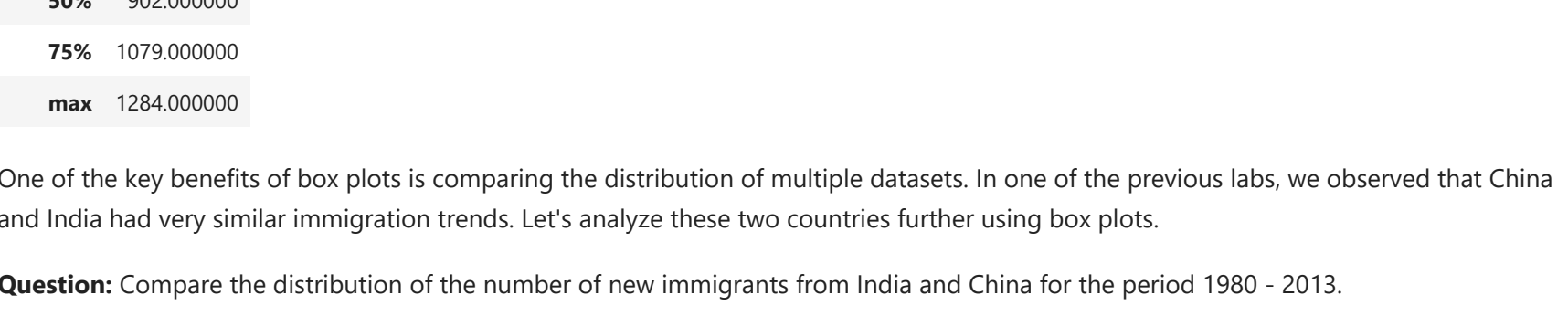


► Click here for a sample python solution

Box Plots

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

- **Minimum:** The smallest number in the dataset excluding the outliers.
- **First quartile:** Middle number between the minimum and the median.
- **Second quartile (Median):** Middle number of the (sorted) dataset.
- **Third quartile:** Middle number between median and maximum.
- **Maximum:** The largest number in the dataset excluding the outliers.



To make a `boxplot`, we can use `kind=box` in `plot()` method invoked on a `pandas` series or dataframe.

Let's plot the box plot for the Japanese immigrants between 1980 - 2013.

```
In [78]: df_can.head()
```

```
Out [78]:
```

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

5 rows × 38 columns

Step 1: Get the subset of the dataset. Even though we are extracting the data for just one country, we will obtain it as a dataframe. This will help us with calling the `dataframe.describe()` method to view the percentiles.

```
In [12]: # to get a dataframe, place extra square brackets around 'Japan'.
df_japan = df_can.loc[['Japan'], years].transpose()
df_japan.head()
```

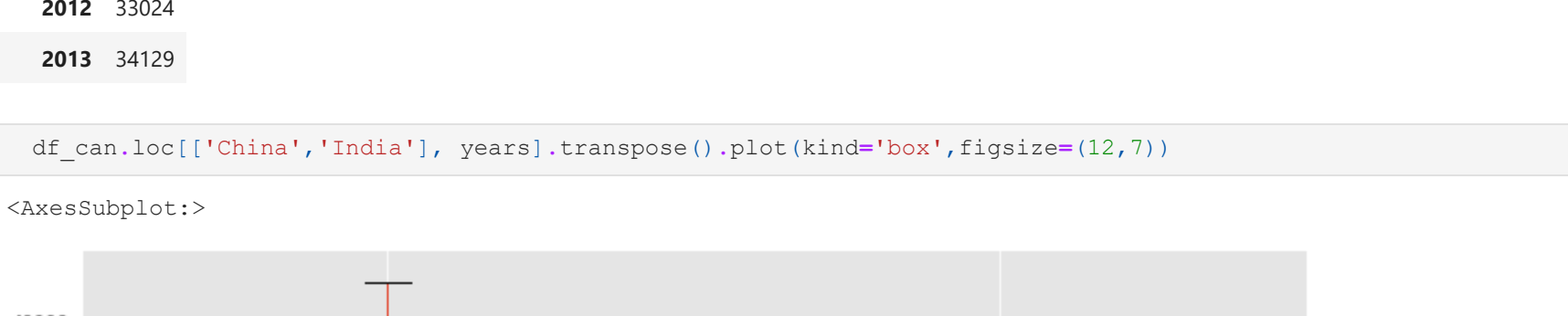
```
Out [12]:
```

	Country	Japan
1980	701	
1981	756	
1982	598	
1983	309	
1984	246	

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

```
In [13]: df_japan.plot(kind='box', figsize=(8, 6))

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



We can immediately make a few key observations from the plot above:

1. The minimum number of immigrants is around 200 (min), maximum number is around 1300 (max), and median number of immigrants is around 900 (median).
2. 25% of the years for period 1980 - 2013 had an annual immigrant count of ~500 or fewer (First quartile).
3. 75% of the years for period 1980 - 2013 had an annual immigrant count of ~1100 or fewer (Third quartile).

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

```
In [14]: df_japan.describe()
```

```
Out [14]:
```

	Country	Japan
count	34.000000	
mean	814.911765	
std	337.219771	
min	198.000000	
25%	529.000000	
50%	902.000000	
75%	1079.000000	
max	1284.000000	

One of the key benefits of box plots is comparing the distribution of multiple datasets. In one of the previous labs, we observed that China and India had very similar immigration trends. Let's analyze these two countries further using box plots.

Question: Compare the distribution of the number of new immigrants from India and China for the period 1980 - 2013.

Step 1: Get the dataset for China and India and call the dataframe `df_ci`.

```
In [28]: ### type your answer here
df_ci = df_can.loc[['China', 'India'], years]
df_ci.shape
```

```
Out [28]: (2, 38)
```

```
In [31]: df_ci.iloc[0:1,3:37].transpose()
```

```
Out [31]:
```

	Country	China	India
Continent	Asia	Asia	
Region	Eastern Asia	Southern Asia	
DevName	Developing regions	Developing regions	
1980	5123	8880	

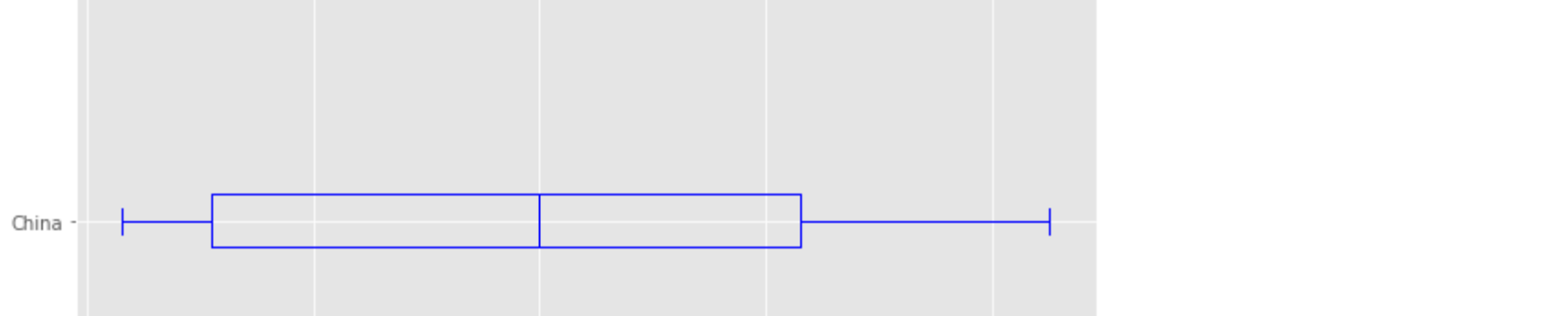
► Click here for a sample python solution

► Click here for a sample python solution

We can observe that, while both countries have around the same median immigrant population (~20,000), China's immigrant population range is more spread out than India's. The maximum population from India for any year (26,210) is around 15% lower than the maximum population from China (42,584).

If you prefer to create horizontal box plots, you can pass the `vert=False` parameter in the `plot` function and assign it to `False`. You can also specify a different color in case you are not a big fan of the default red color.

```
In [45]: # horizontal box plots
df_ci.transpose().iloc[3:37].plot(kind='box', figsize=(10, 7), color='blue', vert=False)
plt.xlabel('Number of Immigrants')
plt.show()
```



Let's view the percentiles associated with both countries using the `describe()` method.

```
In [37]: ### type your answer here
df_ci.describe()
```

```
Out [37]:
```

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010	2011	2012	2013
count	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	...	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000
mean	7001.500000	7676.000000	5727.500000	4600.500000	3615.500000	3013.500000	4555.000000	6416.000000	7140.000000	7333.000000	...	15925.3	14905.4	13345.9	13989.4	14143.4	16384.5	146.0		
std	2656.600177	1405.728281	3421.689714	3871.409627	2953.585025	1693.520741	3669.884194	5335.827771	6197.08383	4256.762823	...	3595.5	3305.3	3349.5	3469.2	3507.8	3342.5	26.0		
min	5123.000000	6682.000000	3308.000000	1983.000000	1527.000000	1816.000000	1960.000000	2643.000000	2758.000000	4323.000000	...	2752.3	2918.8	2828.4	2989.0	3453.4	4089.2	35.0		
25%	6062.250000	7179.000000	4517.750000	3231.750000	2571.250000	2414.750000	3257.500000	4529.500000	4949.000000	5828.000000	...	15925.3	14905.4	13345.9	13989.4	14143.4	16384.5	146.0		
50%	7001.500000	7676.000000	5727.500000	4600.500000	3615.500000	3013.500000	4555.000000	6416.000000	7140.000000	7333.000000	...	15925.3	14905.4	13345.9	13989.4	14143.4	16384.5	146.0		
75%	7940.750000	8175.000000	6937.250000	5969.250000	4659.750000	3612.250000	5852.500000	8302.500000	9331.000000	8838.000000	...	15925.3	14905.4	13345.9						

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010
India	Asia	Southern Asia	Developing regions	8880	8870	8147	7338	5704	4211	7150	...	36210	33848	28742	28261	29456	34235
China	Asia	Eastern Asia	Developing regions	5123	6682	33	1863	1527	1816	1960	...	42584	33518	27642	30037	29622	30391
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
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...	18139	18400	19837	24887	28573	38617
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314	13127	10124	8994	7217	6811
United States of America	Northern America	Northern America	Developed regions	9378	10030	9074	7100	6661	6543	7074	...	8394	9613	9463	10190	8995	8142
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	1172	1429	1822	1592	1977	1648	1794	...	5837	7480	6974	6475	6580	7477
Sri Lanka	Asia	Southern Asia	Developing regions	185	371	290	197	1086	845	1838	...	4930	4714	4123	4756	4547	4422
Republic of Korea	Asia	Eastern Asia	Developing regions	1011	1456	1572	1081	847	962	1208	...	5832	6215	5920	7294	5874	5537
Poland	Europe	Eastern Europe	Developed regions	863	2930	5881	4546	3588	2819	4808	...	1405	1263	1235	1267	1013	795
Lebanon	Asia	Western Asia	Developing regions	1409	1119	1159	789	1253	1683	2576	...	3709	3802	3467	3566	3077	3432
France	Europe	Western Europe	Developed regions	1729	2027	2219	1490	1169	1177	1298	...	4429	4002	4290	4532	5051	4646
Jamaica	Latin America and the Caribbean	Caribbean	Developing regions	3198	2634	2641	2455	2508	2938	4649	...	1948	1722	2141	2334	2456	2321
Viet Nam	Asia	South-Eastern Asia	Developing regions	1191	1829	2162	3404	7583	5907	2741	...	1852	3153	2574	1784	2171	1942
Romania	Europe	Eastern Europe	Developed regions	375	438	583	543	524	604	656	...	5048	4468	3834	2837	2076	1922

15 rows × 38 columns

► Click here for a sample python solution

Step 2: Create a new dataframe which contains the aggregate for each decade. One way to do that:

- Create a list of all years in decades 80's, 90's, and 00's.
- Slice the original dataframe df can to create a series for each decade and sum across all years for each country.
- Merge the three series into a new data frame. Call your dataframe **new_df**.

In [49]:

```
## type your answer here
```

```
#The correct answer is:

# create a list of all years in decades 80's, 90's, and 00's
years_80s = list(map(str, range(1980, 1990)))
years_90s = list(map(str, range(1990, 2000)))
years_00s = list(map(str, range(2000, 2010)))

# slice the original dataframe df can to create a series for each decade
df_80s = df_tot15.loc[:, years_80s].sum(axis=1)
df_90s = df_tot15.loc[:, years_90s].sum(axis=1)
df_00s = df_tot15.loc[:, years_00s].sum(axis=1)

# merge the three series into a new data frame
new_df = pd.DataFrame({'1980s': df_80s, '1990s': df_90s, '2000s': df_00s})

# display dataframe
new_df.head()
```

Out [49]:

	1980s	1990s	2000s
Country	India	82154	180395
	China	32003	161528
	United Kingdom of Great Britain and Northern Ireland	179171	261966
	Philippines	60764	138482
	Pakistan	10591	65302

► Click here for a sample python solution

Let's learn more about the statistics associated with the dataframe using the `describe()` method.

In [50]:

```
## type your answer here
new_df.describe()
```

Out [50]:

	1980s	1990s	2000s
count	15.000000	15.000000	15.000000
mean	44418.333333	85594.666667	97471.533333
std	44190.676455	68237.560246	100583.204205
min	7613.000000	30028.000000	13629.000000
25%	16698.000000	39259.000000	36101.500000
50%	30638.000000	56915.000000	65794.000000
75%	59183.000000	104451.000000	105505.500000
max	179171.000000	261966.000000	340385.000000

► Click here for a sample python solution

Step 3: Plot the box plots.

In [55]:

```
## type your answer here
```

```
fig,ax= plt.subplots(1,3,figsize=(12,6))

new_df['1980s'].plot(kind='box',ax=ax[0])
new_df['1990s'].plot(kind='box',ax=ax[1])
new_df['2000s'].plot(kind='box',ax=ax[2])
```

Out [55]:

<AxesSubplot>

► Click here for a sample python solution

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

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

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

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

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

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

Outlier > 209,611.5

In [61]:

```
new_df=new_df.reset_index()
new_df[new_df['2000s']> 209611.5]
```

Out [61]:

	Country	1980s	1990s	2000s
0	India	82154	180395	303591
1	China	32003	161528	340385

► Click here for a sample python solution

China and India are both considered as outliers since their population for the decade exceeds 209,611.5.

The box plot is an advanced visualization tool, and there are many options and customizations that exceed the scope of this lab. Please refer to [Matplotlib documentation](#) on box plots for more information.

Scatter Plots

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

Let's start by exploring the following:

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

Step 2: Get the dataset. Since we are expecting to use the relationship between **years** and **total population**, we will convert **years** to **int** type.

In [62]:

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

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

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

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

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

Out [62]:

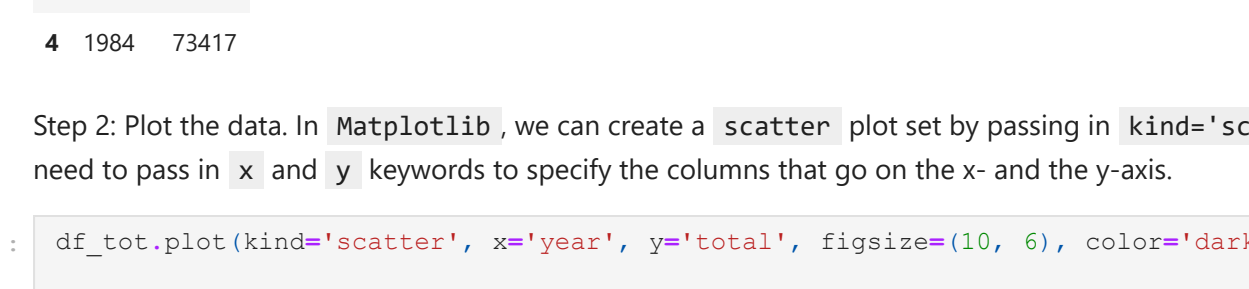
	year	total
0	1980	99137
1	1981	110563
2	1982	104271
3	1983	75550
4	1984	73417

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

In [63]:

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

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



Notice how the scatter plot does not connect the data points together. We can clearly observe an upward trend in the data: as the years go by, the total number of immigrants increases. We can mathematically analyze this upward trend using a regression line (line of best fit). So let's try to plot a linear line of best fit, and use it to predict the number of immigrants in 2015.

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

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

Out [64]:

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

fit

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

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

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

In [65]:

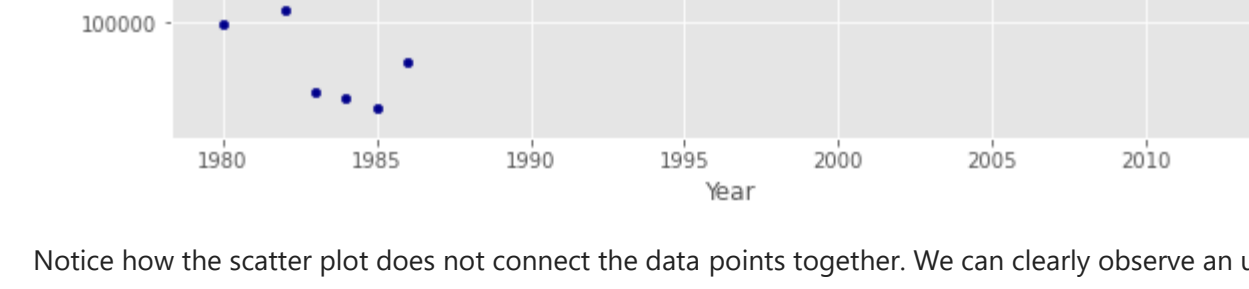
```
df_tot.plot(kind='scatter', x='year', y='total', figsize=(10, 6), color='darkblue')

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

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

plt.show()

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



Out [65]: 'No. Immigrants = 5567 * Year + -10926195'

In [81]:

```
df = df_can.loc[['Denmark', 'Norway', 'Sweden'], years].transpose()
```

In [82]:

```
df.head()
```

Out [82]:

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

Question: Create a scatter plot of the total immigration from Denmark, Norway, and Sweden to Canada from 1980 to 2013?

In [85]:

```
df_total = pd.DataFrame(df.sum(axis=1))
df_total.reset_index(inplace=True)
df_total.columns=['years','total']
```

► Click here for a sample python solution

Step 2: Generate the scatter plot by plotting the total versus year in **df_total**.

In [90]:

```
## type your answer here
```

```
df_total.plot(kind='scatter',x='years',y='total',figsize=(20,8))
```

Out [90]:

<AxesSubplot>



► Click here for a sample python solution

Bubble Plots

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

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

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

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

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

In [91]:

```
# transposed dataframe
df_can_t = df_can[years].transpose()

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

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

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

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

Out [91]:

Country	Year	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Antigua and Barbuda	Argentina	Armenia	...	United States of America	Uruguay	Uzbekistan
0	1980	16	1	80	0	0	1	0	368	0	...	9378	128	0
1	1981	39	0	67	1	0	3	0	426	0	...	10030	122	0
2	1982	39	0	71	0	0	6	0	626	0	...	9074	146	0
3	1983	47	0	69	0	0	6	0	241	0	...	7100	105	0
4	1984	71	0	63	0	0	4	42	237	0	...	6661	90	0

5 rows × 196 columns

Step 2: Create the normalized weights.

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

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

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

In [92]:

```
# normalize Brazil data
norm_brazil = (df_can_t['Brazil'] - df_can_t['Brazil'].min()) / (df_can_t['Brazil'].max() - df_can_t['Brazil'].min())

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

Step 3: Plot the data.

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

In [93]:

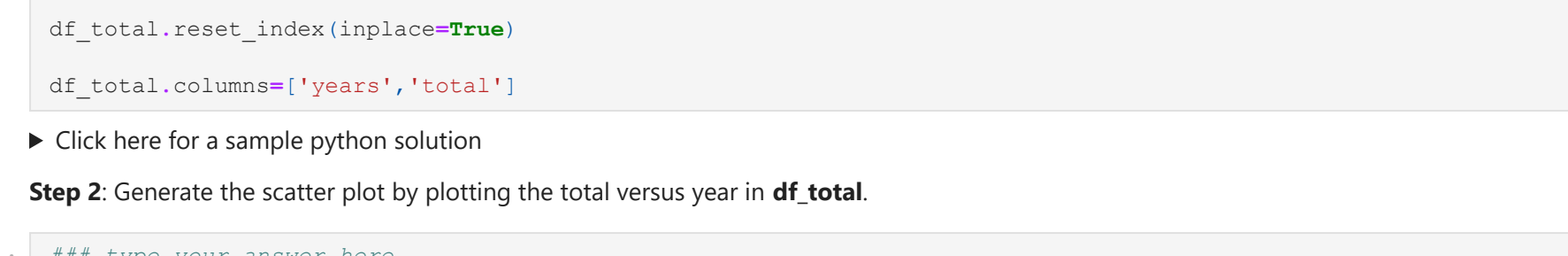
```
# Brazil
ax0 = df_can_t.plot(kind='scatter',
                    x='Year',
                    y='Brazil',
                    figsize=(14, 8),
                    alpha=0.5, # transparency
                    color='green',
                    s=norm_brazil * 2000 + 10, # pass in weights
      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

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

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

Out [93]:

<matplotlib.legend.Legend at 0x7f977d43cf58>



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

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

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

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

Step 1: Normalize the data pertaining to China and India.

In [94]:

```
## type your answer here
```

```
# normalized Chinese data
norm_china = (df_can_t['China'] - df_can_t['China'].min()) / (df_can_t['China'].max() - df_can_t['China'].min())

# normalized Indian data
norm_india = (df_can_t['India'] - df_can_t['India'].min()) / (df_can_t['India'].max() - df_can_t['India'].min())
```

► Click here for a sample python solution

Step 2: Generate the bubble plots.

In [97]:

```
## type your answer here
```

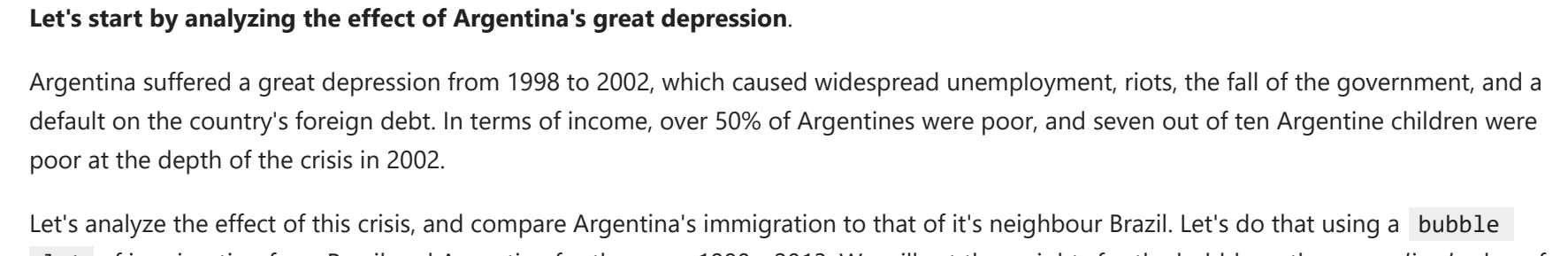
```
ax0 = df_can_t.plot(kind='scatter',
                    x='Year',
                    y='China',
                    figsize=(14, 8),
                    alpha=0.5, # transparency
                    color='yellow',
                    s=norm_china * 2000 + 10, # pass in weights
                    xlim=(1975, 2015)
                    )

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

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

Out [97]:

<matplotlib.legend.Legend at 0x7f977d212a58>



► Click here for a sample python solution

Thank you for completing this lab!

Author

Alex Aklson

Other Contributors

Jay Rajsekharan, Ehsan M. Kermani, Slobodan Markovic, Weiqing Wang.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-05-29	2.6	Weiqing Wang	Fixed typos and code smells.
2021-01-20	2.5	LakshmiHolla	Changed TOC markdown section.
2021-01-05	2.4	LakshmiHolla	Changed markdown for outliers
2020-11-12	2.3	LakshmiHolla	Added example code for outliers
2020-11-03	2.2	LakshmiHolla	Changed URL of excel file
2020-09-29	2.1	LakshmiHolla	Made fix to a boxplot label
2020-08-27	2.0	Lavanja	Moved lab to course repo in GitHub