UNIT – IV

Introduction to Matplotlib

Matplotlib is the most popular plotting library for python which gives control over every aspect of a figure. It was designed to give the end user a similar feeling like MATLAB's graphical plotting. In the coming sections we will learn about Seaborn that is built over matplotlib. The official page of Matplotlib is https://matplotlib.org. You can use this page for official installation instructions and various documentation links. One of the most important section on this page is the gallery section - https://matplotlib.org/gallery.html - it shows all the kind of plots/figures that matplotlib is capable of creating for you. You can select anyone of those, and it takes you the example page having the figure and very well documented code. Another important page is https://matplotlib.org/api/pyplot_summary.html- and it has the documentation functions in it.

Matplotlib's architecture is composed of three main layers: the back-end layer, the artist layer where much of the heavy lifting happens and is usually the appropriate programming paradigm when writing a web application server, or a UI application, or perhaps a script to be shared with other developers, and the scripting layer, which is the appropriate layer for everyday purposes and is considered a lighter scripting interface to simplify common tasks and for a quick and easy generation of graphics and plots.

Now let's go into each layer in a little more detail:

Back-end layer has three built-in abstract interface classes: FigureCanvas, which defines and encompasses the area on which the figure is drawn. Renderer, an instance of the renderer class knows how to draw on the figure canvas. And finally, Event, which handles user inputs such as keyboard strokes and mouse clicks.

Artist layer: It 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. There are two types of Artist objects. The first type is the primitive type, such as a line, a rectangle, a circle, or text. And the second type is the composite type, such as the figure or the axes. The top-level Matplotlib object that contains and manages all of the elements in a given graphic is the figure Artist, and the most important composite artist is the axes because it is where most of the Matplotlib API plotting methods are defined, including methods to create and manipulate the ticks, the axis lines, the grid or the plot background. Now it is important to note that each composite artist may contain other composite artists as well as primitive artists. So, a figure artist for example would contain an axis artist as well as a rectangle or text artists.

Scripting layer: it was developed for scientists who are not professional Programmers. The 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, which automates the process of defining a canvas and defining a figure artist instance and connecting them.

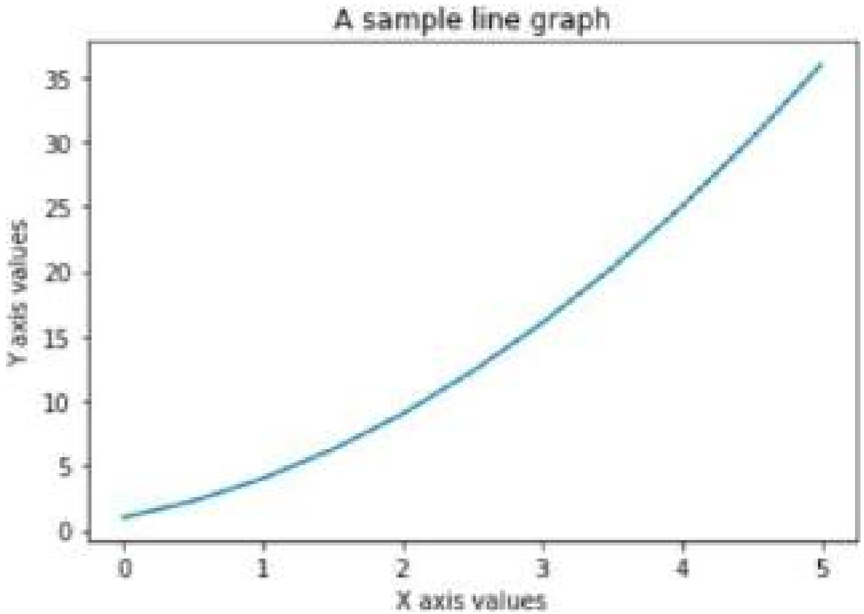
Read a CSV and Generate a Line Plot with Matplotlib

A line plot is used to represent quantitative values over a continuous interval or time period. It is generally used to depict trends on how the data has changed over time.

In this sub-section, we will see how to use matplotlib to read a csv file and then generate a plot. We will use jupyter notebook. First, we do a basic example to showcase what a line plot is.

```
In [1]: import matplotlib.pyplot as plt

        #### The below line will allow you to view the plot inside the jupiter notebook

In [2]: %matplotlib inline

In [3]: import numpy as np
        x = np.linspace(0,5,11)
        y = (x+1) **2

In [4]: x

Out[4]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])

In [5]: y

Out[5]: array([ 1.  ,  2.25,  4.  ,  6.25,  9.  , 12.25, 16.  , 20.25, 25.  ,
               30.25, 36.  ])
```

```
        #### Now we will plot the graph for these two variables

In [9]: plt.plot(x,y)
        plt.xlabel('X axis values')
        plt.ylabel('Y axis values')
        plt.title('A sample line graph')

Out[9]: Text(0.5, 1.0, 'A sample line graph')
```



Now let us do a small case study using what we just learned now:

Download the dataset from the link: https://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.asp
The data set has all the country immigration information. We will use the one for Australia for our case study.

Now let us use this to plot immigration data

```
import numpy as np
import pandas as pd
```

We need to insall xlrd module that pandas need to read excel files. If you are using anaconda distribution then you can do it by using the command :

conda install -c anaconda xlrd --yes

```
df_aus = pd.read_excel('Australia.xlsx',sheet_name='Australia by Residence',skiprows=range(20),skipfooter=2)
print("pandas dataframe contains this data now")
```

pandas dataframe contains this data now

```
df_aus.head()
```

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2004	2005	2006	2007	2008	2009	2010	2011	2012
0	Emigrants	Both	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	0	...	80	120	70	80	120	...	...	...	...
1	Emigrants	Both	Albania	908	Europe	925	Southern Europe	901	Developed regions	0	...	30	40	30	30	30	...	...	...	...
2	Emigrants	Both	Algeria	903	Africa	912	Northern Africa	902	Developing regions	20	...	20	20	30	40	30	...	...	...	...
3	Emigrants	Both	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	10	...	0	0	0	0	0	...	...	...	...
4	Emigrants	Both	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	...	0	0	0	0	0	...	...	...	...

5 rows × 43 columns

```
df_aus.tail()
```

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2004	2005	2006	2007	2008	2009	2010	2011	2012
441	Immigrants	Both	Wallis and Futuna Islands	909	Oceania	957	Polynesia	902	Developing regions	0	...	10	0	0	0	0	...	...	...	...
442	Immigrants	Both	Western Sahara	903	Africa	912	Northern Africa	902	Developing regions	0	...	0	0	0	0	0	...	...	...	...
443	Immigrants	Both	Yemen	935	Asia	922	Western Asia	902	Developing regions	10	...	40	20	10	40	40	...	...	...	...
444	Immigrants	Both	Zambia	903	Africa	910	Eastern Africa	902	Developing regions	150	...	370	240	410	410	400	...	...	...	...
445	Immigrants	Both	Zimbabwe	903	Africa	910	Eastern Africa	902	Developing regions	830	...	1750	1580	1880	2160	2270	...	...	...	...

5 rows × 43 columns

Get the list of column names and the list of indices

```
df_aus.columns.values
```

```
array(['Type', 'Coverage', 'OdName', 'AREA', 'AreaName', 'REG', 'RegName',
       'DEV', 'DevName', 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987,
       1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998,
       1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009,
       2010, 2011, 2012, 2013], dtype=object)
```

```
df_aus.index.values
```

```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
       156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
       169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
       182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
```

Use the tolist() method to get index and columns as lists. View the dimensions of dataframe using “.shape” parameter.

After that let us clean the data set to remove few unnecessary columns.

```
df_aus.columns.tolist()
df_aus.index.tolist()

print (type(df_aus.columns.tolist()))
print (type(df_aus.index.tolist()))
```

```
<class 'list'>
<class 'list'>
```

```
df_aus.shape
```

```
(446, 43)
```

```
#let's clean the data set to remove a few unnecessary columns. We can use pandas drop() method as follows:
df_aus.drop(['AREA','REG','DEV','Type','Coverage'],axis=1, inplace=True)
df_aus.head(2)
```

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	Afghanistan	Asia	Southern Asia	Developing regions	0	0	0	0	0	0	...	80	120	70	30	120	...	...	...	...	...
1	Albania	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	...	30	40	30	30	30	...	...	...	...	...

2 rows × 38 columns

```
df_aus.describe()
```

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989
count	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000
mean	871.547085	858.704753	842.757340	787.025112	767.443946	758.560538	843.529171	920.717489	1024.753363	1050.022422
std	5282.000001	5381.755219	5331.664256	5320.463642	4953.103474	5034.944088	5156.875347	5487.016488	6054.203458	5910.235594
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	10.000000	10.000000	10.000000	10.000000	16.000000	16.000000	10.000000	10.000000	18.000000	10.000000
75%	130.000000	140.000000	150.000000	150.000000	160.000000	156.000000	177.500000	186.000000	248.000000	227.500000
max	80860.000000	85580.000000	82340.000000	100610.000000	85360.000000	83448.000000	82460.000000	87770.000000	104778.000000	120040.000000

8 rows × 30 columns

Let us rename the column names so that it makes more sense.

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

```
Index([  'Country', 'Continent',    'Region',   'DevName',        1980,
               1981,        1982,        1983,        1984,        1985,
               1986,        1987,        1988,        1989,        1990,
               1991,        1992,        1993,        1994,        1995,
               1996,        1997,        1998,        1999,        2000,
               2001,        2002,        2003,        2004,        2005,
               2006,        2007,        2008,        2009,        2010,
               2011,        2012,        2013,     'Total'],
      dtype='object')
```

```
df_aus.describe()
```

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989
count	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000	446.000000
mean	811.547085	850.784753	842.757848	787.526112	787.443946	795.560538	843.520179	920.717489	1024.753363	1056.022422
std	8282.800004	8361.755216	8321.854250	8328.403142	8059.803474	8034.944099	8156.576347	8487.016408	8094.283458	8010.228594
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
75%	130.000000	140.000000	150.000000	150.000000	160.000000	160.000000	177.500000	190.000000	240.000000	227.500000
max	90040.000000	95680.000000	92340.000000	108510.000000	98390.000000	93442.000000	92450.000000	97770.000000	124770.000000	120040.000000

8 rows × 39 columns

```
df_sus.head(3)
```

	Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
0	Afghanistan	Asia	Southern Asia	Developing regions	0	0	0	0	0	0	...	120	70	80	120	...	...	...	...	...	1090
1	Albania	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	...	40	30	30	30	...	...	...	...	...	1410
2	Algeria	Africa	Northern Africa	Developing regions	20	10	10	10	10	10	...	30	30	40	50	...	...	...	...	...	1170

3 rows × 39 columns

Default index is numerical, but it is more convenient to index based on country names.

```
df_sus.set_index('Country', inplace=True)
df_sus.head(3)
```

	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	0	0	0	0	0	0	0	...	120	70	80	120	...	...	...	...	...	1090
Albania	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0	...	40	30	30	30	...	...	...	...	...	1410
Algeria	Africa	Northern Africa	Developing regions	20	10	10	10	10	10	0	...	20	30	40	50	...	...	...	...	...	1170

3 rows × 38 columns

Remove the name of the index.

```
df_sus.index.name = None
```

```
df_sus.head(3)
```

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Afghanistan	Asia	Southern Asia	Developing regions	0	0	0	0	0	0	0	...	120	70	80	120	...	...	...	...	...	1090
Albania	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	0	...	40	30	30	30	...	...	...	...	...	1410
Algeria	Africa	Northern Africa	Developing regions	20	10	10	10	10	10	0	...	20	30	40	50	...	...	...	...	...	1170

3 rows × 38 columns

Let us now test it by pulling the data for Bangladesh.

```
print(df_aus[df_aus.index == 'Bangladesh'].T.squeeze())
```

```
                     Bangladesh          Bangladesh
Continent                  Asia                Asia
Region            Southern Asia       Southern Asia
DevName      Developing regions  Developing regions
1980                        220                 170
1981                        140                 180
1982                        170                 220
1983                        180                 140
1984                        160                 170
1985                        140                 220
1986                        140                 190
1987                        140                 150
1988                        110                 270
1989                        180                 270
1990                        130                 400
1991                        230                 600
1992                        200                 690
1993                        190                 360
1994                        180                 690
1995                        190                1050
1996                        210                 900
```

Column names as numbers could be confusing. For example: year 1985 could be misunderstood as 1985th column. To avoid ambiguity, let us convert column names to strings and then use that to call full range of years.

```
df_aus.columns = list(map(str,df_aus.columns))
```

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

```
['1980',
 '1981',
 '1982',
 '1983',
 '1984',
 '1985',
 '1986',
 '1987',
 '1988',
 '1989',
 '1990',
 '1991',
 '1992',
 '1993',
 '1994',
 '1995',
 '1996',
 '1997',
```

We can also pass multiple criteria in the same line.

```
df_aus[(df_aus['Continent']=='Asia') & (df_aus['Region']=='Southern Asia')]
```

	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	3	0	3	0	9	3	0	...	120	75	80	120	...	...	...	...	...	
Bangladesh	Asia	Southern Asia	Developing regions	220	140	170	180	160	140	140	...	540	480	530	530	...	...	...	...	...	2
Bhutan	Asia	Southern Asia	Developing regions	10	10	10	0	8	3	10	...	50	80	70	100	...	...	...	...	...	
India	Asia	Southern Asia	Developing regions	740	660	750	780	779	740	760	...	4510	4530	4930	5260	...	...	...	...	...	16
Iran (Islamic Republic of)	Asia	Southern Asia	Developing regions	40	30	30	30	29	50	60	...	100	390	340	350	...	...	...	...	...	2
Maldives	Asia	Southern Asia	Developing regions	10	10	10	0	16	20	10	...	60	110	70	70	...	...	...	...	...	
Nepal	Asia	Southern Asia	Developing regions	40	70	50	40	60	100	120	...	220	180	180	280	...	...	...	...	...	1
[illegible]	Asia	Southern Asia	Developing	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]						[illegible]

Let us review the changes we have made to our dataframes.

```
print('data dimensions:', df_aus.shape)
print(df_aus.columns)
df_aus.head(2)
```

```
data dimensions: (446, 38)
Index(['Continent', 'Region', 'DevName', '1980', '1981', '1982', '1983',
       '1984', '1985', '1986', '1987', '1988', '1989', '1990', '1991', '1992',
       '1993', '1994', '1995', '1996', '1997', '1998', '1999', '2000', '2001',
       '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '2010',
       '2011', '2012', '2013', 'Total'],
      dtype='object')
```

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Afghanistan	Asia	Southern Asia	Developing regions	3	6	6	0	9	9	3	...	120	70	10	120	...	...	...	...	...	1690
Albania	Europe	Southern Europe	Developed regions	3	6	6	3	6	6	3	...	40	30	10	30	...	...	...	...	...	1410

2 rows × 38 columns

Case Study – let us now study the trend of number of immigrants from Bangladesh to Australia.

Case Study - Do a plot of the immigrants from bangladesh

```
bangladesh = df_aus.loc['Bangladesh',years]
bangladesh.head()
```

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
Bangladesh	220	140	170	180	160	140	140	140	110	180	...	480	540	480	500	530	...	...	...	...	...
Bangladesh	170	180	200	140	170	220	150	150	270	270	...	2470	1860	2570	1080	3160	...	...	...	...	...

2 rows × 34 columns

Since there are two rows of data, let us sum the values of each column and take first 20 years (to eliminate other years for which no values are present).

```
pd_bang=bangladesh.sum(axis=0,skipna=True).head(20)
```

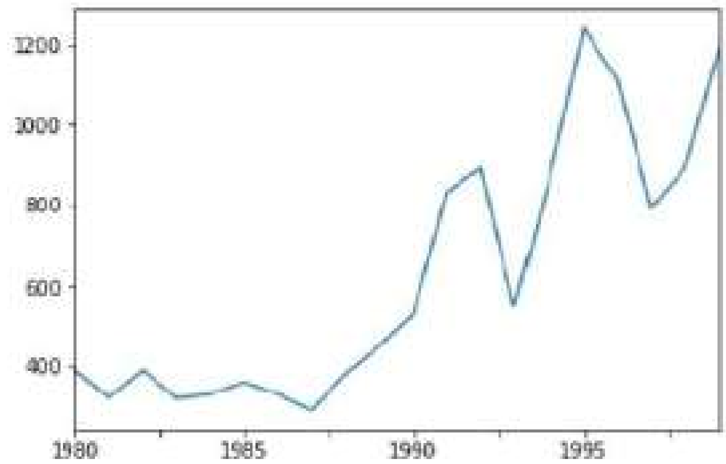
```
pd_bang
```

```
1980     390
1981     320
1982     390
1983     320
1984     330
1985     360
1986     330
1987     290
1988     380
1989     450
1990     530
1991     830
1992     890
1993     550
1994     870
1995    1240
1996    1110
1997     790
1998     890
```

Next, we can plot by using the plot function. Automatically the x-axis is plotted with the index values and y-axis with column values

```
pd_bang.plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0xc8485c0>



Basic Plots using Matplotlib
Area Plot

In the previous module we used line plot to see immigration from Bangladesh to Australia. Now let us try different types of basic plotting using matplotlib.

Area plot

Now let us use area plots to see to visualize cumulative immigration from top 5 countries to Canada. We will use the same process to clean data that we used in the previous section.

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

URL - https://s3-api.us-geo.objectstorage.softlayer.net/cf-coursesdata/CognitiveClass/DV0101EN/labs/Data_Files/Canada.xlsx

```
df_can = pd.read_excel('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data_F
                       sheet_name='Canada by Citizenship',
                       skiprows=range(20),
                       skipfooter=2
                      )

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

```
Data downloaded and read into a dataframe!
```

```
df_can.head()
```

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

5 rows × 43 columns

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

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

```
(195, 43)
```

Now clean up data using the same process as the one in the previous section :

```python
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()
```

	OdName	AreaName	RegName	DevName	1980	1981	1982	1983	1984	1985	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	1450	1223	856	702	560	716	561	539	620	603
2	Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	...	3616	3626	4807	3623	4005	5393	4752	4325	3774	4331
3	American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0
4	Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	...	0	0	1	1	0	0	0	0	1	1

5 rows × 38 columns

```python
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()
```

	Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	1450	1223	856	702	560	716	561	539	620	603
2	Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	...	3616	3626	4807	3623	4005	5393	4752	4325	3774	4331
3	American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0
4	Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	...	0	0	1	1	0	0	0	0	1	1

5 rows × 38 columns

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

False

```python
df_can.columns = list(map(str, df_can.columns))

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

True

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

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

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1450	1223	856	702	560	716	561	539	620	603
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3616	3626	4807	3623	4005	5393	4752	4325	3774	4331
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2	...	0	0	1	1	0	0	0	0	1	1

5 rows × 37 columns

```python
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()
```

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
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

5 rows × 38 columns

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

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

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

years
```

```
['1980',
 '1981',
 '1982',
 '1983',
 '1984',
 '1985',
 '1986',
 '1987',
 '1988'
```

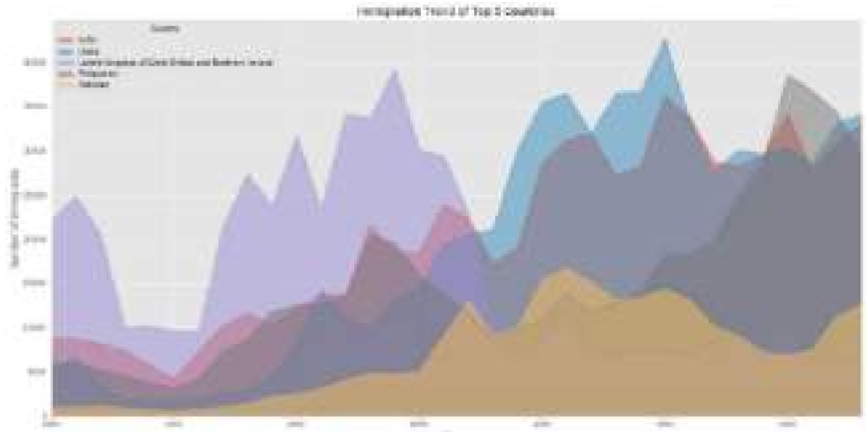
```python
# use the inline backend to generate the plots within the browser
%matplotlib inline

import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.style.use('ggplot') # optional: for ggplot-like style

# check for latest version of Matplotlib
print ('Matplotlib version: ', mpl.__version__) # >= 2.0.0
```

```
Matplotlib version:  3.0.3
```

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

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

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

df_top5.head()
```

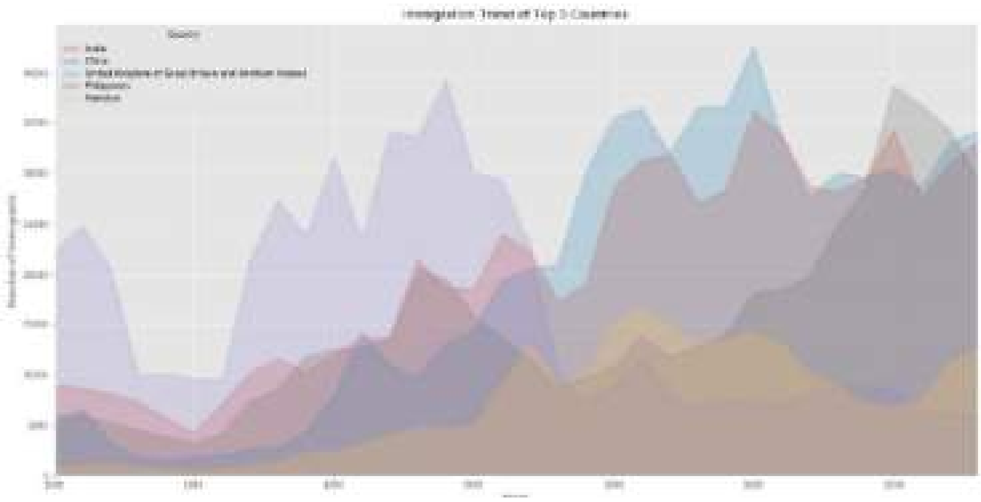
Country	India	China	United Kingdom of Great Britain and Northern Ireland	Philippines	Pakistan
1980	8880	5123	22045	6051	978
1981	8670	6682	24796	5921	972
1982	8147	3308	20620	5249	1201
1983	7338	1863	10015	4562	900
1984	5704	1527	10170	3801	668

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

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

plt.show()
```



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

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

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

plt.show()
```



Bar Chart

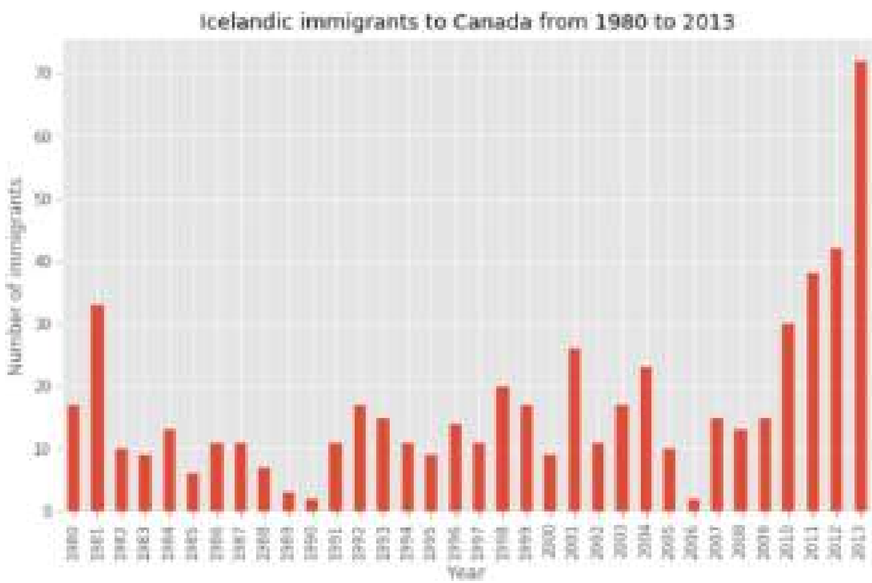
A bar plot is a way of representing data where the length of the bars represents the magnitude/size of the feature/variable. Bar graphs usually represent numerical and categorical variables grouped in intervals.

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

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

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

```python
# 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()
```



Histogram

How could you visualize the answer to the following question ?

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

To answer this one would need to plot a histogram - 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.

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

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

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

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

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

By default, the histogram 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:

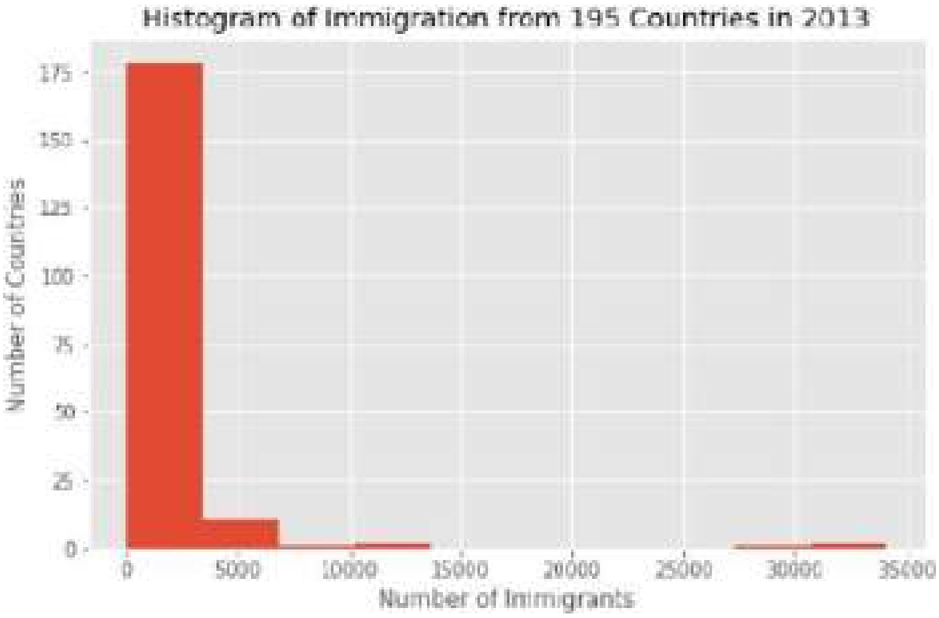
178 Countries contributed between 0 to 3412.9 immigrants

11 Countries contributed between 3412.9 to 6825.8 immigrants

1 Country contributed between 6285.8 to 10238.7 immigrants, and so on.

	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Bin 6	Bin 7	Bin 8	Bin 9	Bin 10
Range	0 to 3412.9	3412.9 to 6825.8	6825.8 to 10238.7	10238.7 to 13651.6	13651.6 to 17064.5	17064.5 to 20477.4	20477.4 to 23890.3	23890.3 to 27303.2	27303.2 to 30716.1	30716.1 to 34129.
Frequency	178	11	1	2	0	0	0	0	1	2

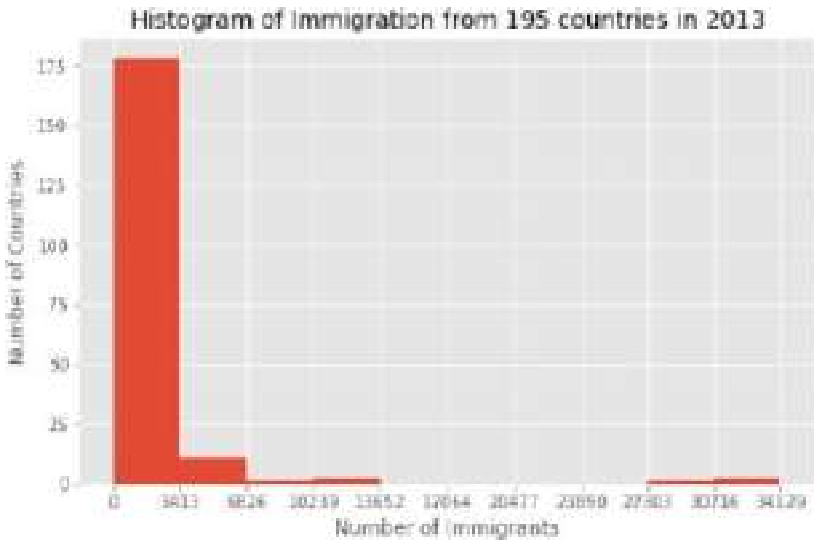
```
df_can['2013'].plot(kind='hist', figsize=(8, 5))

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

plt.show()
```



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



Specialized Visualization Tools using Matplotlib

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. We can continue to use the same dataframe further.

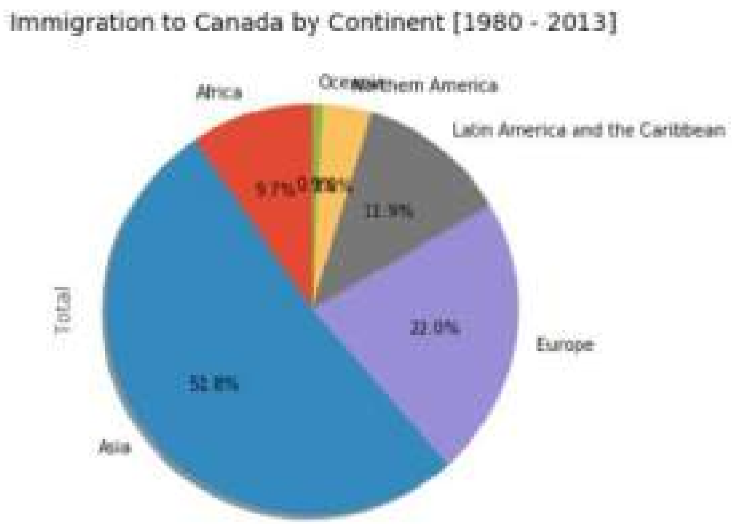
```
# group countries by continents and apply sum() function
df_continents = df_can.groupby('Continent', axis=0).sum()

# 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_can.groupby('Continent', axis=0)))

df_continents.head()
```

```
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

Continent	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010	2011	2012
Africa	3951	4363	3819	2671	2639	2650	3782	7494	7552	9894	...	27523	29188	28284	29890	34534	40892	35441	38083
Asia	31025	34314	30214	24696	27274	23850	28739	43203	47454	60256	...	159253	149054	133459	139894	141434	163845	146894	152218
Europe	39760	44802	42720	24638	22287	20844	24370	46698	54726	60893	...	35955	33053	33495	34692	35078	33425	26778	29177
Latin America and the Caribbean	13081	15215	16769	15427	13678	15171	21179	28471	21924	25060	...	24747	24676	26011	26547	26867	28818	27856	27173
Northern America	9378	10030	9074	7100	6661	6543	7074	7705	6469	6790	...	8394	9613	9463	10190	8995	8142	7677	7892

5 rows × 35 columns

```
# autopct create %, start angle represent starting point
df_continents['Total'].plot(kind='pie',
                            figsize=(5, 6),
                            autopct='%1.1f%%', # add in percentages
                            startangle=90,     # start angle 90° (Africa)
                            shadow=True,       # add shadow
                            )

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:

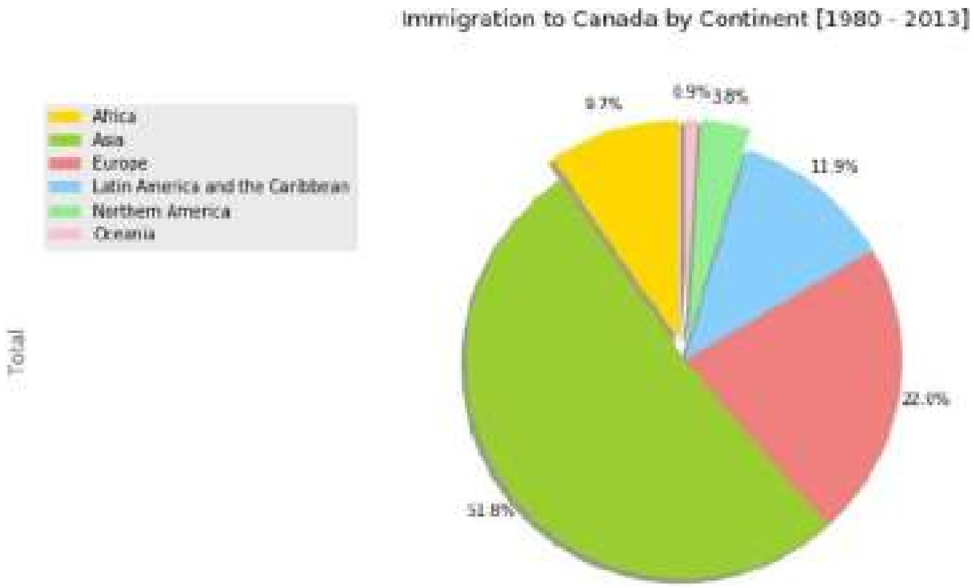
```
colors_list = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink']
explode_list = [0.1, 0, 0, 0, 0.1, 0.1] # ratio for each continent with which to offset each wedge.

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 of the text g
                            colors=colors_list,  # add custom colors
                            explode=explode_list # 'explode' lowest 3 continents
                            )

# scale the title up by 12% 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()
```

Raw code :

colors_list = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink'] explode_list = [0.1, 0, 0, 0, 0.1, 0.1] # ratio for each continent with which to offset each wedge.

df_continents['Total'].plot(kind='pie',
figsize=(15, 6),
autopct='%1.1f%%',
startangle=90,
shadow=True,
labels=None, # turn off labels on pie chart

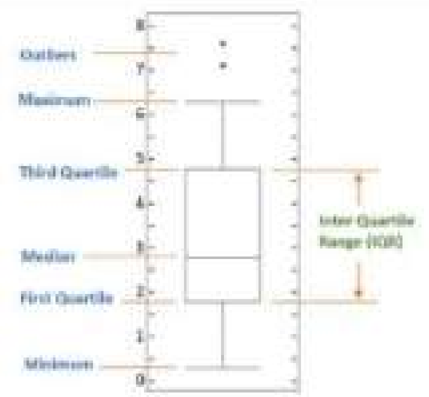
```
pctdistance=1.12, # the ratio between the center of each pie slice and the start of the text generated by autopct
colors=colors_list, # add custom colors
explode=explode_list # 'explode' lowest 3 continents)

# scale the title up by 12% 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()
```



Box Plot

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

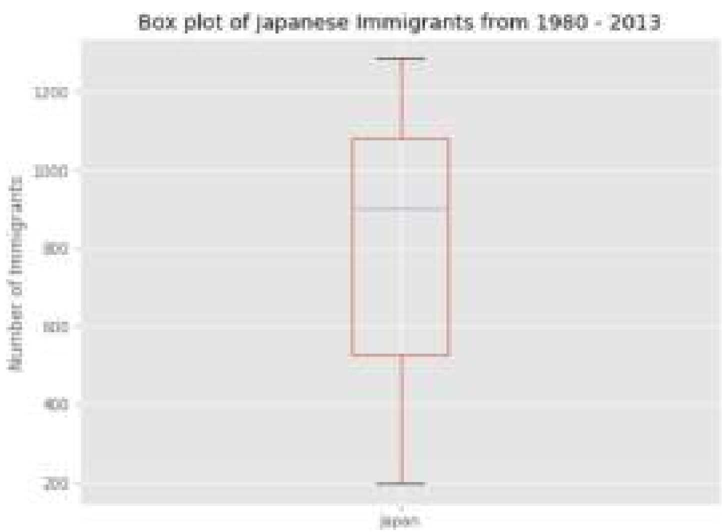
Minimum: Smallest number in the dataset.
First quartile: Middle number between the minimum and the median.
Second quartile (Median): Middle number of the (sorted) dataset.
Third quartile: Middle number between median and maximum.
Maximum: Highest number in the dataset.




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

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

```
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:

The minimum number of immigrants is around 200 (min), maximum number is around 1300 (max), and median number of immigrants is around 900 (median).

25% of the years for period 1980 - 2013 had an annual immigrant count of ~500 or fewer (First quartile).

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.

```
df_japan.describe()
```

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

Scatter Plots

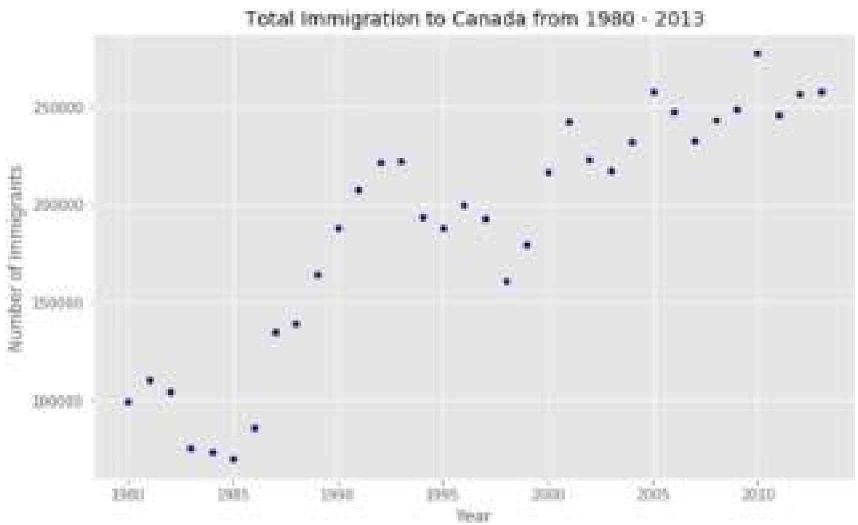
A scatter plot (2D) is a useful method of comparing variables against each other. Scatter plots look similar to line plots in that they both map independent and dependent variables on a 2D graph. While the datapoints are connected by a line in a line plot, they are not connected in a scatter plot. The 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.

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

```python
# 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()
```

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

```python
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()
```



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.

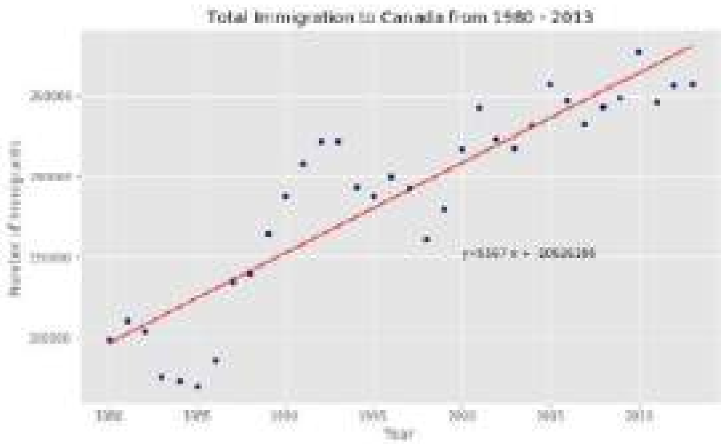
deg: Degree of fitting polynomial. 1 = linear, 2 = quadratic, and so on.

```
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])
```

Plot the regression line on the scatter plot.

```
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])
```



'No. Immigrants = 5567 * Year + -10926195'

Bubble Plots

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

Let us compare Argentina's immigration to that of its neighbor 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.

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

# 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()
```

Country	Year	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Antigua and Barbuda	Argentina	Armenia	...	United States of America	Uruguay	Uzbekistan	Vanuatu	Venezuela (Bolivarian Republic of)
0	1980	16	1	80	0	0	1	0	368	0	...	9378	128	0	0	103
1	1981	39	0	67	1	0	3	0	426	0	...	10030	132	0	0	117
2	1982	39	0	71	0	0	6	0	626	0	...	9074	146	0	0	174
3	1983	47	0	69	0	0	6	0	241	0	...	7100	105	0	0	124
4	1984	71	0	63	0	0	4	42	237	0	...	6661	90	0	0	142

5 rows × 196 columns

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:

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

```
# 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']
```

Raw Code :

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()

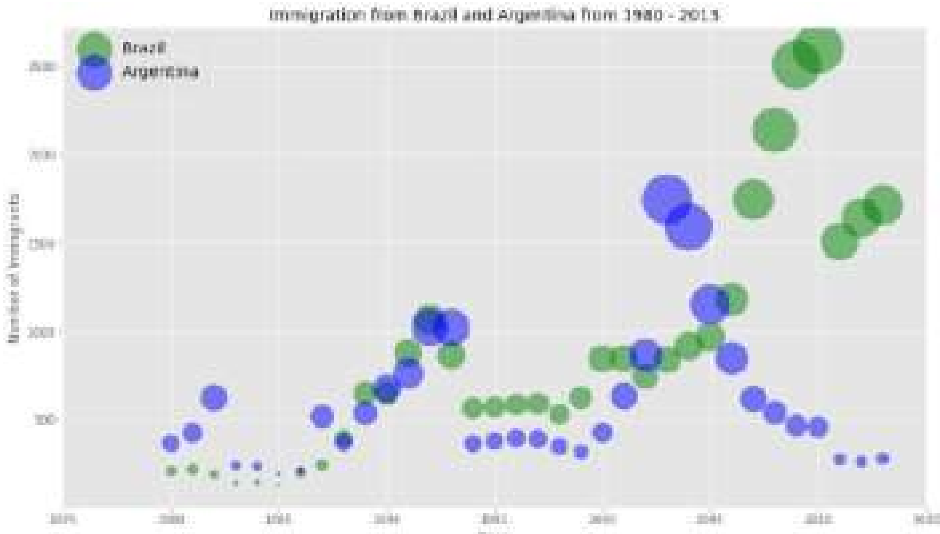
```
# Brazil
ax0 = df_can_t.plot(kind='scatter',
                    x='Year',
                    y='Brazil',
                    figsize=(14, 8),
                    alpha=0.5,                  # transparency
                    color='green',
                    s=norm_brazil * 2000 + 10,  # pass 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 - 2013')
ax0.legend(['Brazil', 'Argentina'], loc='upper left', fontsize='x-large')
```

Raw Code :

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 - 2013')
ax0.legend(['Brazil', 'Argentina'], loc='upper left', fontsize='x-large')
```



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, the more immigrants in that year.

Waffle Chart

A waffle chart is an interesting visualization that is normally created to display progress toward goals. It is commonly an effective option when you are trying to add interesting visualization features to a visual that consists mainly of cells, such as an Excel dashboard.

```python
# let's create a new dataframe for these three countries
df_dsn = df_can.loc[['Denmark', 'Norway', 'Sweden'], :]

# let's take a look at our dataframe
df_dsn
```

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Country																					
Denmark	Europe	Northern Europe	Developed regions	272	293	299	106	93	73	93	...	62	101	97	108	81	92	93	94	81	3901
Norway	Europe	Northern Europe	Developed regions	116	77	106	51	31	54	56	...	57	53	73	66	75	46	49	53	59	2327
Sweden	Europe	Northern Europe	Developed regions	281	308	222	176	128	158	187	...	205	139	193	165	167	159	134	140	140	5866

The first step into creating a waffle chart is determing the proportion of each category with respect to the total.

```python
# compute the proportion of each category with respect to the total
total_values = sum(df_dsn['Total'])
category_proportions = [(float(value) / total_values) for value in df_dsn['Total']]

# print out proportions
for i, proportion in enumerate(category_proportions):
    print (df_dsn.index.values[i] + ': ' + str(proportion))
```

```
Denmark: 0.32255663965602777
Norway: 0.1924094592359848
Sweden: 0.48503390110798744
```

The second step is defining the overall size of the waffle chart.

```python
width = 40 # width of chart
height = 10 # height of chart

total_num_tiles = width * height # total number of tiles

print ('Total number of tiles is ', total_num_tiles)
```

```
Total number of tiles is  400
```

The third step is using the proportion of each category to determe it respective number of tiles

```
# compute the number of tiles for each category
tiles_per_category = [round(proportion * total_num_tiles) for proportion in category_proportions]

# print out number of tiles per category
for i, tiles in enumerate(tiles_per_category):
    print (df_dsn.index.values[i] + ': ' + str(tiles))
```

```
Denmark: 129
Norway: 77
Sweden: 194
```

The fourth step is creating a matrix that resembles the waffle chart and populating it.

```
# initialize the waffle chart as an empty matrix
waffle_chart = np.zeros((height, width))

# define indices to loop through waffle chart
category_index = 0
tile_index = 0

# populate the waffle chart
for col in range(width):
    for row in range(height):
        tile_index += 1

        # if the number of tiles populated for the current category is equal to its corresponding allocated tiles...
        if tile_index > sum(tiles_per_category[0:category_index]):
            # ...proceed to the next category
            category_index += 1

        # set the class value to an integer, which increases with class
        waffle_chart[row, col] = category_index

print ('Waffle chart populated!')
```

```
Waffle chart populated!
```

Raw Code :

# initialize the waffle chart as an empty matrix
waffle_chart = np.zeros((height, width))

# define indices to loop through waffle chart
category_index = 0
tile_index = 0

# populate the waffle chart
for col in range(width):
for row in range(height):
tile_index += 1

# if the number of tiles populated for the current category is equal to its corresponding allocated tiles...
if tile_index > sum(tiles_per_category[0:category_index]):

```
        # ...proceed to the next category
        category_index += 1

    # set the class value to an integer, which increases with class
    waffle_chart[row, col] = category_index


print ('Waffle chart populated!')
```

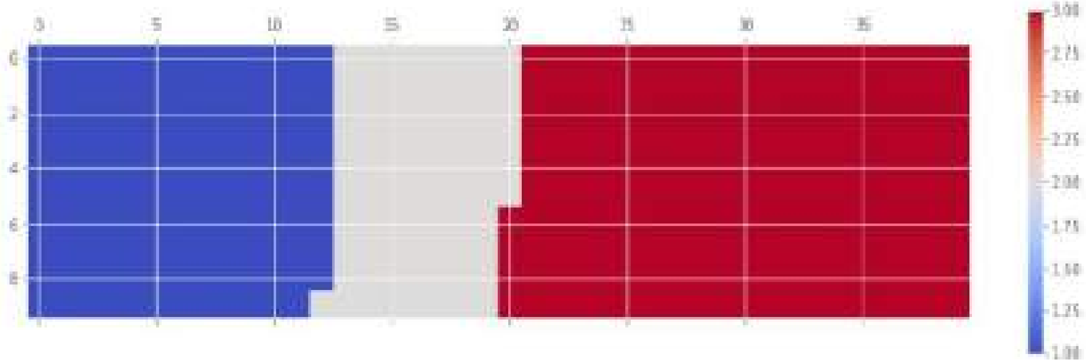
```
waffle_chart
```

```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2.,
        2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2., 2.,
        2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
        3., 3., 3., 3., 3., 3., 3., 3.]])
```

Map the waffle chart matrix into a visual.

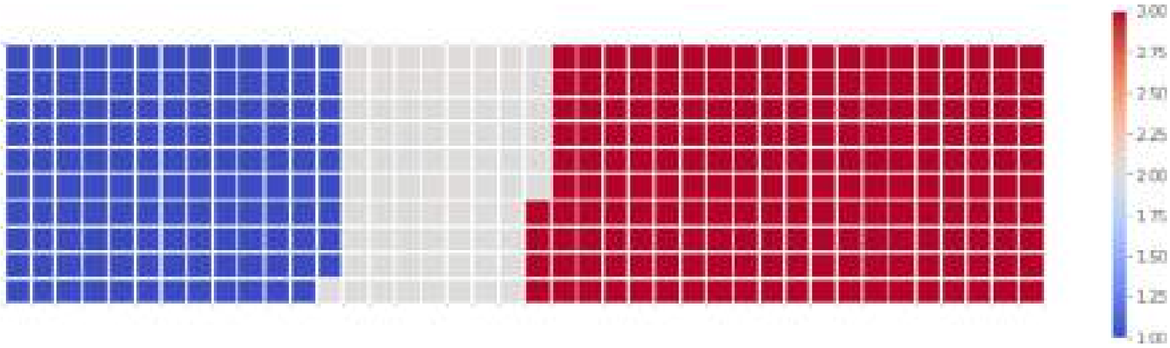
```
# instantiate a new figure object
fig = plt.figure()

# use matshow to display the waffle chart
colormap = plt.cm.coolwarm
plt.matshow(waffle_chart, cmap=colormap)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x4076cc0>

<Figure size 432x288 with 0 Axes>
```



Prettify the chart.

```
# instantiate a new figure object
fig = plt.figure()

# use matshow to display the waffle chart
colormap = plt.cm.coolwarm
plt.matshow(waffle_chart, cmap=colormap)
plt.colorbar()

# get the axis
ax = plt.gca()

# set minor ticks
ax.set_xticks(np.arange(-.5, (width), 1), minor=True)
ax.set_yticks(np.arange(-.5, (height), 1), minor=True)

# add gridlines based on minor ticks
ax.grid(which='minor', color='w', linestyle='-', linewidth=2)

plt.xticks([])
plt.yticks([])
```



Word Clouds

Word clouds (also known as text clouds or tag clouds) work in a simple way: the more a specific word appears in a source of textual data (such as a speech, blog post, or database), the bigger and bolder it appears in the word cloud.

```
# import package and its set of stopwords
from wordcloud import WordCloud, STOPWORDS

print ('Wordcloud is installed and imported!')
```

```
Wordcloud is installed and imported!
```

```
# open the file and read it into a variable alice_novel
alice_novel = open('alice_novel.txt', 'r').read()

print ('File downloaded and saved!')
```

```
File downloaded and saved!
```

```
stopwords = set(STOPWORDS)
```

```
# instantiate a word cloud object
alice_wc = WordCloud(
    background_color='white',
    max_words=2000,
    stopwords=stopwords
)

# generate the word cloud
alice_wc.generate(alice_novel)
```

```
# display the word cloud
plt.imshow(alice_wc, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Interesting! So, in the first 2000 words in the novel, the most common words are Alice, said, little, Queen, and so on. Let's resize the cloud so that we can see the less frequent words a little better.

```
fig = plt.figure()
fig.set_figwidth(14) # set width
fig.set_figheight(18) # set height

# display the cloud
plt.imshow(alice_wc, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Much better! However, said isn't really an informative word. So, let's add it to our stop words and re-generate the cloud.

```
stopwords.add('said') # add the words said to stopwords

# re-generate the word cloud
alice_wc.generate(alice_novel)

# display the cloud
fig = plt.figure()
fig.set_figwidth(14) # set width
fig.set_figheight(18) # set height

plt.imshow(alice_wc, interpolation='bilinear')
plt.axis('off')
plt.show()
```