(https://cognitiveclass.ai)

Waffle Charts, Word Clouds, and Regression Plots

In [ ]:

```
#by Christopher Harrison
```

Introduction

In this lab, we will learn how to create word clouds and waffle charts. Furthermore, we will start learning about additional visualization libraries that are based on Matplotlib, namely the library *seaborn*, and we will learn how to create regression plots using the *seaborn* library.

Table of Contents

Exploring Datasets with *pandas* and Matplotlib

Toolkits: The course heavily relies on *pandas* (http://pandas.pydata.org/) and **Numpy** (http://www.numpy.org/) for data wrangling, analysis, and visualization. The primary plotting library we will explore in the course is Matplotlib (http://matplotlib.org/).

Dataset: Immigration to Canada from 1980 to 2013 - International migration flows to and from selected countries - The 2015 revision (http://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.shtml) 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 [1]:

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 to read in excel files. This module is **xlrd**. For your convenience, we have pre-installed this module, so you would not have to worry about that. Otherwise, you would need to run the following line of code to install the **xlrd** module:

```
!conda install -c anaconda xlrd --yes
```

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

In [2]:

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

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

In [3]:

Out[3]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	198
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	1
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	8
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	

5 rows × 43 columns

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

In [4]:

```
(195, 43)
```

Clean up data. We will make some modifications to the original dataset to make it easier to create our visualizations. Refer to *Introduction to Matplotlib and Line Plots* and *Area Plots, Histograms, and Bar Plots* for a detailed description of this preprocessing.

In [5]:

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

Visualizing Data using Matplotlib

Import `matplotlib` :

In [6]:

```
Matplotlib version:  3.3.3
```

Waffle Charts

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.

Let's revisit the previous case study about Denmark, Norway, and Sweden.

In [7]:

Out[7]:

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005
Denmark	Europe	Northern Europe	Developed regions	272	293	299	106	93	73	93	...	62
Norway	Europe	Northern Europe	Developed regions	116	77	106	51	31	54	56	...	57
Sweden	Europe	Northern Europe	Developed regions	281	308	222	176	128	158	187	...	205

3 rows × 38 columns

Unfortunately, unlike R, `waffle` charts are not built into any of the Python visualization libraries. Therefore, we will learn how to create them from scratch.

Step 1. The first step into creating a waffle chart is determing the proportion of each category with respect to the total.

In [8]:

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

Step 2. The second step is defining the overall size of the `waffle` chart.

In [9]:

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

Step 3. The third step is using the proportion of each category to determe it respective number of tiles

In [10]:

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

Based on the calculated proportions, Denmark will occupy 129 tiles of the `waffle` chart, Norway will occupy 77 tiles, and Sweden will occupy 194 tiles.

Step 4. The fourth step is creating a matrix that resembles the `waffle` chart and populating it.

In [11]:

```
Waffle chart populated!
```

Let's take a peek at how the matrix looks like.

In [12]:

Out[12]:

```
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.]])
```

As expected, the matrix consists of three categories and the total number of each category's instances matches the total number of tiles allocated to each category.

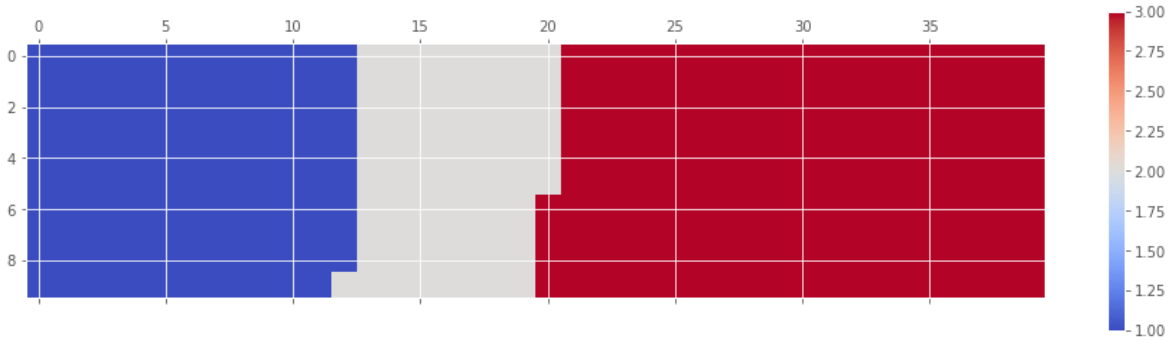
Step 5. Map the `waffle` chart matrix into a visual.

In [13]:

Out[13]:

```
<matplotlib.colorbar.Colorbar at 0x7f35ba064898>
```

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



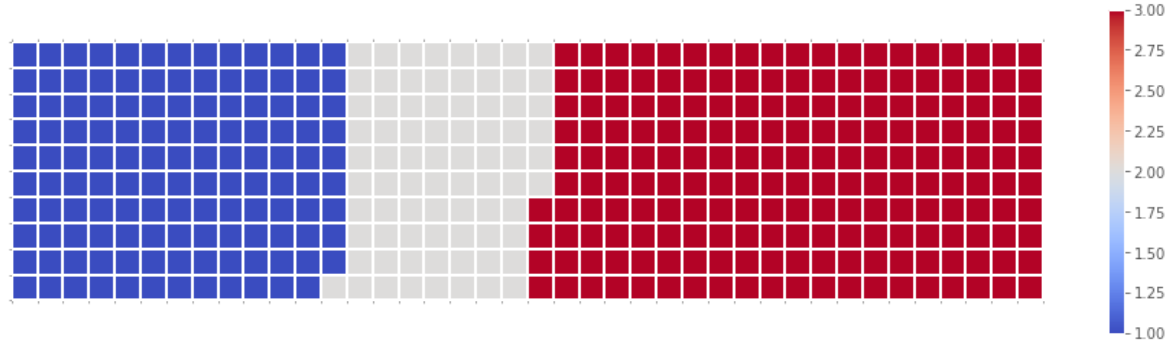
Step 6. Prettify the chart.

In [14]:

Out[14]:

```
([], [])
```

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



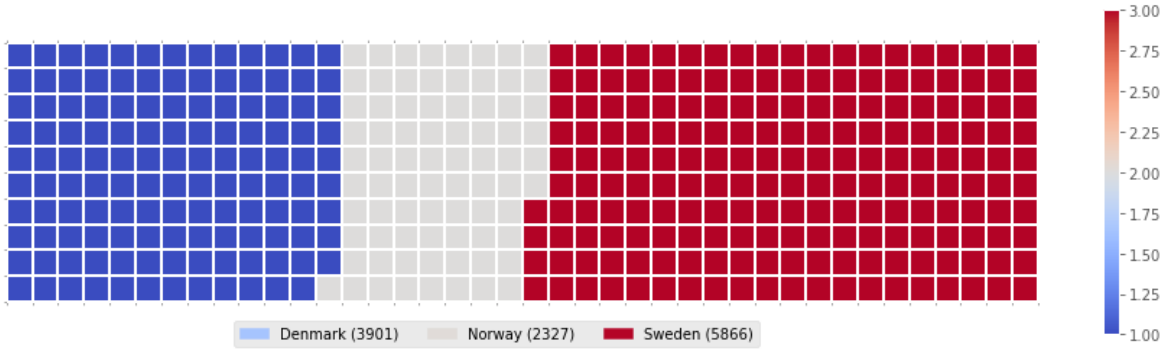
Step 7. Create a legend and add it to chart.

In [15]:

Out[15]:

```
<matplotlib.legend.Legend at 0x7f35b1cd6e80>
```

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



And there you go! What a good looking *delicious* `waffle` chart, don't you think?

Now it would very inefficient to repeat these seven steps every time we wish to create a `waffle` chart. So let's combine all seven steps into one function called *create_waffle_chart*. This function would take the following parameters as input:

1. **categories**: Unique categories or classes in dataframe.
2. **values**: Values corresponding to categories or classes.
3. **height**: Defined height of waffle chart.
4. **width**: Defined width of waffle chart.
5. **colormap**: Colormap class
6. **value_sign**: In order to make our function more generalizable, we will add this parameter to address signs that could be associated with a value such as %, $, and so on. **value_sign** has a default value of empty string.

In [16]:

Now to create a `waffle` chart, all we have to do is call the function `create_waffle_chart` . Let's define the input parameters:

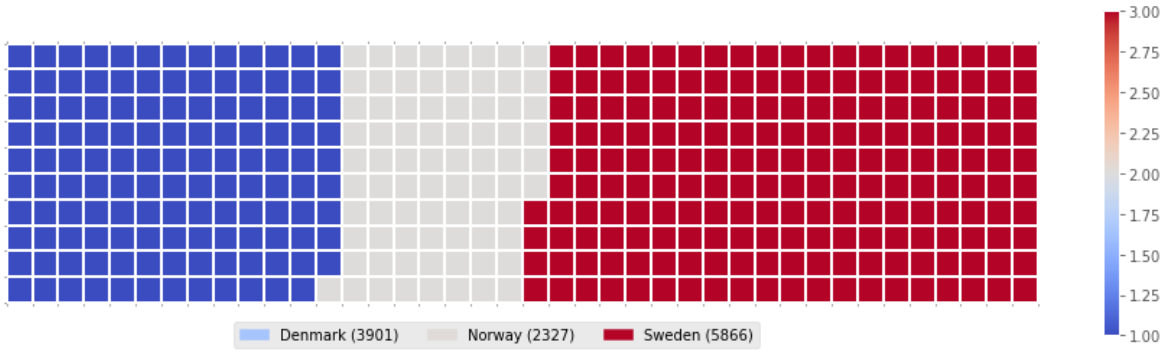
In [17]:

And now let's call our function to create a `waffle` chart.

In [18]:

```
Total number of tiles is 400
Denmark: 129
Norway: 77
Sweden: 194

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



There seems to be a new Python package for generating `waffle charts` called PyWaffle (https://github.com/ligyxy/PyWaffle), but it looks like the repository is still being built. But feel free to check it out and play with it.

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.

Luckily, a Python package already exists in Python for generating `word` clouds. The package, called `word_cloud` was developed by **Andreas Mueller**. You can learn more about the package by following this link (https://github.com/amueller/word_cloud/).

Let's use this package to learn how to generate a word cloud for a given text document.

First, let's install the package.

In [19]:

```
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with f
lexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda/envs/python

  added / updated specs:
    - wordcloud==1.4.1


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    wordcloud-1.4.1            |           py36_0         324 KB  cond
a-forge
    ------------------------------------------------------------
                                           Total:         324 KB

The following NEW packages will be INSTALLED:

  wordcloud          conda-forge/linux-64::wordcloud-1.4.1-py36_0

The following packages will be UPDATED:

  certifi            pkgs/main::certifi-2020.12.5-py36h06a~ --> conda-
forge::certifi-2020.12.5-py36h5fab9bb_1

The following packages will be SUPERSEDED by a higher-priority channe
l:

  ca-certificates    pkgs/main::ca-certificates-2020.12.8-~ --> conda-
forge::ca-certificates-2020.12.5-ha878542_0
  openssl              pkgs/main::openssl-1.1.1i-h27cfd23_0 --> conda-
forge::openssl-1.1.1i-h7f98852_0



Downloading and Extracting Packages
wordcloud-1.4.1      | 324 KB    | ####################################
## | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Wordcloud is installed and imported!
```

`Word` clouds are commonly used to perform high-level analysis and visualization of text data. Accordingly, let's digress from the immigration dataset and work with an example that involves analyzing text data. Let's try to analyze a short novel written by **Lewis Carroll** titled *Alice's Adventures in Wonderland*. Let's go ahead and download a *.txt* file of the novel.

In [20]:

```
File downloaded and saved!
```

Next, let's use the stopwords that we imported from `word_cloud` . We use the function *set* to remove any redundant stopwords.

In [21]:

Create a word cloud object and generate a word cloud. For simplicity, let's generate a word cloud using only the first 2000 words in the novel.

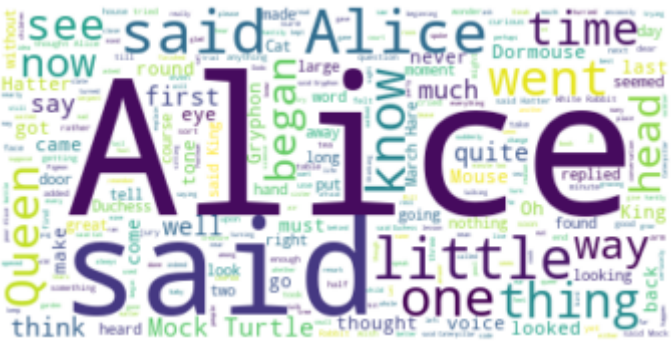
In [22]:

Out[22]:

```
<wordcloud.wordcloud.WordCloud at 0x7f3648cac2e8>
```

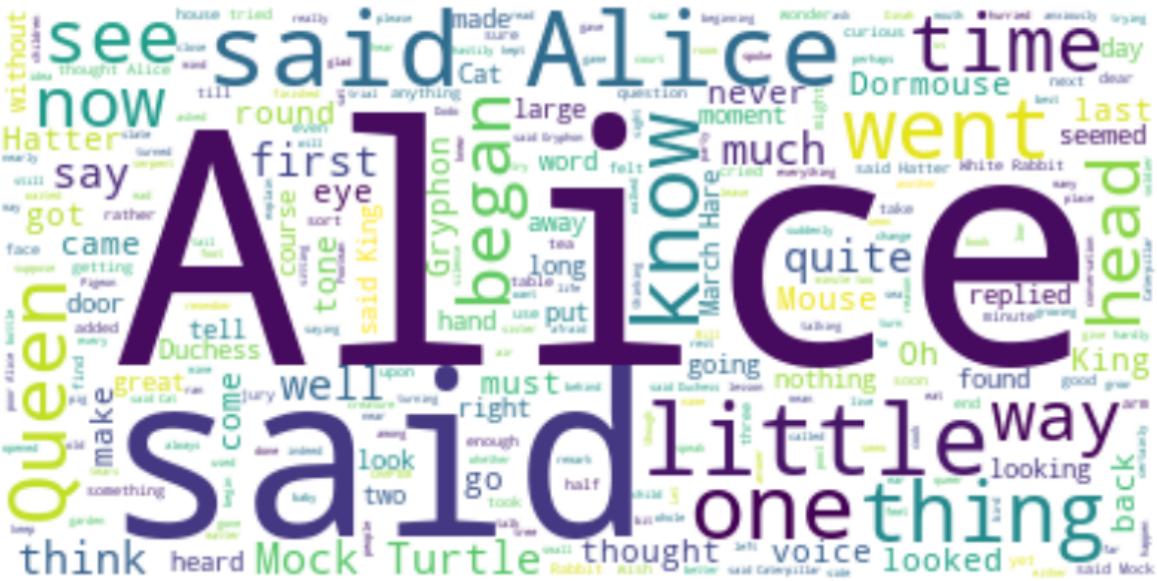
Awesome! Now that the `word` cloud is created, let's visualize it.

In [23]:



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.

In [24]:



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

In [25]:

Excellent! This looks really interesting! Another cool thing you can implement with the `word_cloud` package is superimposing the words onto a mask of any shape. Let's use a mask of Alice and her rabbit. We already created the mask for you, so let's go ahead and download it and call it *alice_mask.png*.

In [26]:

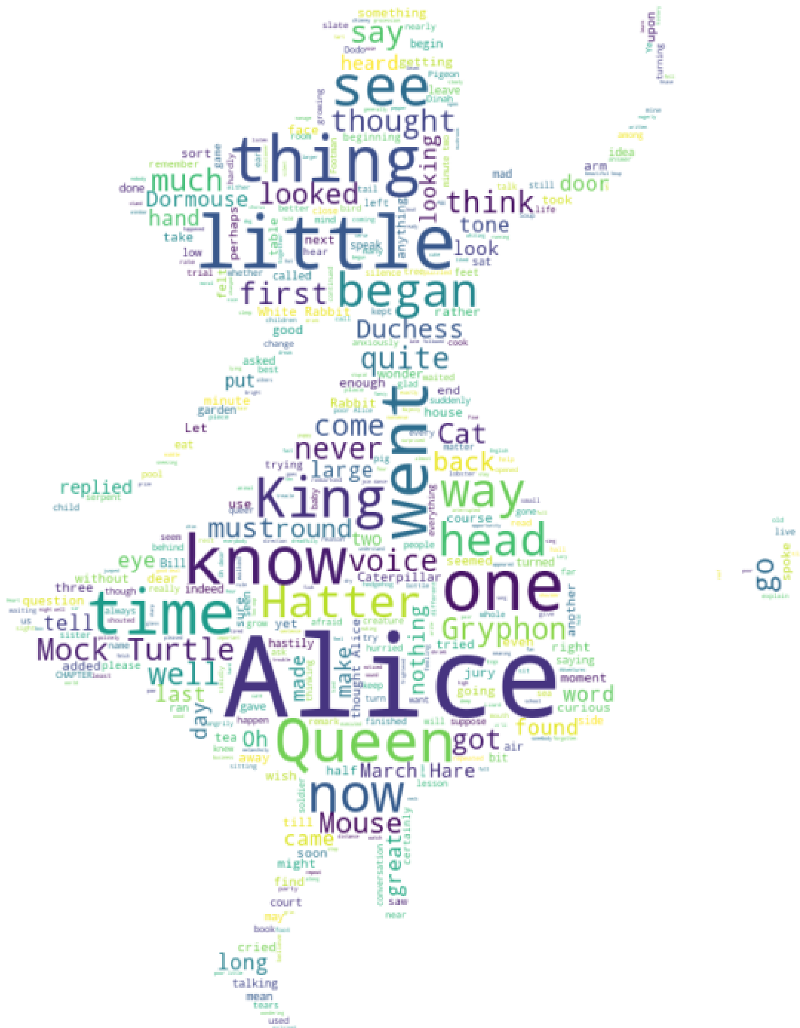
```
Image downloaded and saved!
```

Let's take a look at how the mask looks like.

In [27]:

Shaping the `word` cloud according to the mask is straightforward using `word_cloud` package. For simplicity, we will continue using the first 2000 words in the novel.

In [28]:



Really impressive!

Unfortunately, our immmigration data does not have any text data, but where there is a will there is a way. Let's generate sample text data from our immigration dataset, say text data of 90 words.

Let's recall how our data looks like.

In [29]:

Out[29]:

Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	3
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	1
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	3
American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0	...	
Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2	...	

5 rows × 38 columns

And what was the total immigration from 1980 to 2013?

In [30]:

Out[30]:

```
6409153
```

Using countries with single-word names, let's duplicate each country's name based on how much they contribute to the total immigration.

In [31]:

Out[31]:

```
'China China China China China China China China China Colombia Egypt
France Guyana Haiti India India India India India India India India In
dia Jamaica Lebanon Morocco Pakistan Pakistan Pakistan Philippines Phi
lippines Philippines Philippines Philippines Philippines Philippines P
oland Portugal Romania '
```

We are not dealing with any stopwords here, so there is no need to pass them when creating the word cloud.

In [32]:

```
Word cloud created!
```

In [33]:

According to the above word cloud, it looks like the majority of the people who immigrated came from one of 15 countries that are displayed by the word cloud. One cool visual that you could build, is perhaps using the map of Canada and a mask and superimposing the word cloud on top of the map of Canada. That would be an interesting visual to build!

Regression Plots

> Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics. You can learn more about *seaborn* by following this link (https://seaborn.pydata.org/) and more about *seaborn* regression plots by following this link (http://seaborn.pydata.org/generated/seaborn.regplot.html).

In lab *Pie Charts, Box Plots, Scatter Plots, and Bubble Plots*, we learned how to create a scatter plot and then fit a regression line. It took ~20 lines of code to create the scatter plot along with the regression fit. In this final section, we will explore *seaborn* and see how efficient it is to create regression lines and fits using this library!

Let's first install *seaborn*

In [ ]:

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda/envs/python

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    blas-1.0                   |              mkl           6 KB  anac
onda
    ca-certificates-2020.10.14 |                0         128 KB  anac
onda
    certifi-2020.6.20          |           py36_0         160 KB  anac
onda
    dbus-1.13.16               |       hb2f20db_0         589 KB  anac
onda
    gst-plugins-base-1.14.0    |       hbbd80ab_1         4.8 MB
    gstreamer-1.14.0           |       h28cd5cc_2         3.2 MB
    libgfortran-ng-7.3.0       |       hdf63c60_0         1.3 MB  anac
onda
    matplotlib-3.3.1           |                0          24 KB  anac
onda
    matplotlib-base-3.3.1      |   py36h817c723_0         6.7 MB  anac
onda
    mkl-service-2.3.0          |   py36he8ac12f_0          52 KB
    mkl_fft-1.2.0              |   py36h23d657b_0         149 KB
    mkl_random-1.1.1           |   py36h0573a6f_0         382 KB  anac
onda
    numpy-1.19.2               |   py36h54aff64_0          22 KB
    numpy-base-1.19.2          |   py36hfa32c7d_0         4.1 MB
    openssl-1.1.1h             |       h7b6447c_0         3.8 MB  anac
onda
    pyqt-5.9.2                 |   py36h22d08a2_1         5.6 MB  anac
onda
    qt-5.9.7                   |       h5867ecd_1        68.5 MB
    scipy-1.5.2                |   py36h0b6359f_0        14.4 MB
    seaborn-0.11.0             |             py_0         216 KB  anac
onda
    sip-4.19.24                |   py36he6710b0_0         297 KB  anac
onda
    ------------------------------------------------------------
                                           Total:       114.4 MB

The following NEW packages will be INSTALLED:

  blas               anaconda/linux-64::blas-1.0-mkl
  dbus               anaconda/linux-64::dbus-1.13.16-hb2f20db_0
  gst-plugins-base   pkgs/main/linux-64::gst-plugins-base-1.14.0-hbbd8
0ab_1
  gstreamer          pkgs/main/linux-64::gstreamer-1.14.0-h28cd5cc_2
```

```
  matplotlib          anaconda/linux-64::matplotlib-3.3.1-0
  mkl-service         pkgs/main/linux-64::mkl-service-2.3.0-py36he8ac12
f_0
  mkl_fft             pkgs/main/linux-64::mkl_fft-1.2.0-py36h23d657b_0
  mkl_random          anaconda/linux-64::mkl_random-1.1.1-py36h0573a6f_
0
  numpy-base          pkgs/main/linux-64::numpy-base-1.19.2-py36hfa32c7
d_0
  pyqt                anaconda/linux-64::pyqt-5.9.2-py36h22d08a2_1
  qt                  pkgs/main/linux-64::qt-5.9.7-h5867ecd_1
  scipy               pkgs/main/linux-64::scipy-1.5.2-py36h0b6359f_0
  seaborn             anaconda/noarch::seaborn-0.11.0-py_0
  sip                 anaconda/linux-64::sip-4.19.24-py36he6710b0_0

The following packages will be REMOVED:

  libblas-3.9.0-3_openblas
  libcblas-3.9.0-3_openblas
  libgfortran5-9.3.0-he4bcb1c_17
  liblapack-3.9.0-3_openblas
  libopenblas-0.3.12-pthreads_h4812303_1

The following packages will be SUPERSEDED by a higher-priority channe
l:

  ca-certificates    conda-forge::ca-certificates-2020.12.~ --> anacon
da::ca-certificates-2020.10.14-0
  certifi            conda-forge::certifi-2020.12.5-py36h5~ --> anacon
da::certifi-2020.6.20-py36_0
  libgfortran-ng     conda-forge::libgfortran-ng-9.3.0-he4~ --> anacon
da::libgfortran-ng-7.3.0-hdf63c60_0
  matplotlib-base    conda-forge::matplotlib-base-3.3.3-py~ --> anacon
da::matplotlib-base-3.3.1-py36h817c723_0
  numpy              conda-forge::numpy-1.19.4-py36h2aa4a0~ --> pkgs/m
ain::numpy-1.19.2-py36h54aff64_0
  openssl            conda-forge::openssl-1.1.1i-h7f98852_0 --> anacon
da::openssl-1.1.1h-h7b6447c_0



Downloading and Extracting Packages
seaborn-0.11.0       | 216 KB    | ####################################
## | 100%
scipy-1.5.2          | 14.4 MB   | ####################################
## | 100%
certifi-2020.6.20    | 160 KB    | ####################################
## | 100%
qt-5.9.7             | 68.5 MB   | ####################################
## | 100%
pyqt-5.9.2           | 5.6 MB    | ####################################
## | 100%
blas-1.0             | 6 KB      | ####################################
## | 100%
mkl-service-2.3.0    | 52 KB     | ####################################
## | 100%
dbus-1.13.16         | 589 KB    | ####################################
## | 100%
```

```
mkl_random-1.1.1      | 382 KB    | ####################################
## | 100%
mkl_fft-1.2.0         | 149 KB    | ####################################
## | 100%
matplotlib-3.3.1      | 24 KB     | ####################################
## | 100%
openssl-1.1.1h        | 3.8 MB    | ####################################
## | 100%
matplotlib-base-3.3.  | 6.7 MB    | ####################################
## | 100%
numpy-1.19.2          | 22 KB     | ####################################
## | 100%
libgfortran-ng-7.3.0  | 1.3 MB    | ####################################
## | 100%
sip-4.19.24           | 297 KB    | ####################################
## | 100%
gst-plugins-base-1.1  | 4.8 MB    | ####################################
## | 100%
numpy-base-1.19.2     | 4.1 MB    | ####################################
## | 100%
gstreamer-1.14.0      | 3.2 MB    | ####################################
## | 100%
ca-certificates-2020  | 128 KB    | ####################################
## | 100%
Preparing transaction: done
Verifying transaction: \
```

Create a new dataframe that stores that total number of landed immigrants to Canada per year from 1980 to 2013.

In [ ]:

With *seaborn*, generating a regression plot is as simple as calling the **regplot** function.

In [ ]:

This is not magic; it is *seaborn*! You can also customize the color of the scatter plot and regression line. Let's change the color to green.

In [ ]:

You can always customize the marker shape, so instead of circular markers, let's use '+'.

In [ ]:

Let's blow up the plot a little bit so that it is more appealing to the sight.

In [ ]:

And let's increase the size of markers so they match the new size of the figure, and add a title and x- and y-labels.

In [ ]:

And finally increase the font size of the tickmark labels, the title, and the x- and y-labels so they don't feel left out!

In [ ]:

Amazing! A complete scatter plot with a regression fit with 5 lines of code only. Isn't this really amazing?

If you are not a big fan of the purple background, you can easily change the style to a white plain background.

In [ ]:

Or to a white background with gridlines.

In [ ]:

Question: Use seaborn to create a scatter plot with a regression line to visualize the total immigration from Denmark, Sweden, and Norway to Canada from 1980 to 2013.

In [ ]:

Double-click **here** for the solution.

Thank you for completing this lab!

This notebook was created by Alex Aklson (https://www.linkedin.com/in/aklson/). I hope you found this lab interesting and educational. Feel free to contact me if you have any questions!

This notebook is part of a course on **Coursera** called *Data Visualization with Python*. If you accessed this notebook outside the course, you can take this course online by clicking here (http://cocl.us/DV0101EN_Coursera_Week3_LAB1).
