Waffle Charts, Word Clouds, and Regression Plots

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Create Word cloud and Waffle charts
- Create regression plots with Seaborn library

Table of Contents

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
from PIL import Image # converting images into arrays
```

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

```
! pip3 install openpyxl
```

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

In [3]:
```
df_can = pd.read_excel(
    'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
    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!
```

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

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	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)
```

```
(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 [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':'Regi

# 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
df_can.set_index('Country', inplace = True)

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

# years that we will 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 and setup `matplotlib` :

In [7]:

```
%matplotlib inline

import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches # needed for waffle Charts

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
```

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 [8]:

```
# 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
```

Out[8]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005
Country												
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 [9]:
```
# compute the proportion of each category with respect to the total
total_values = df_dsn['Total'].sum()
category_proportions = df_dsn['Total'] / total_values

# print out proportions
pd.DataFrame({"Category Proportion": category_proportions})
```

Out[9]:

Country	Category Proportion
Denmark	0.322557
Norway	0.192409
Sweden	0.485034

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

In [10]:
```
width = 40 # width of chart
height = 10 # height of chart

total_num_tiles = width * height # total number of tiles

print(f'Total number of tiles is {total_num_tiles}.')
```

```
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 [11]:
```
# compute the number of tiles for each category
tiles_per_category = (category_proportions * total_num_tiles).round().astype(int)

# print out number of tiles per category
pd.DataFrame({"Number of tiles": tiles_per_category})
```

Out[11]:

Country	Number of tiles
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 [12]:
```
# initialize the waffle chart as an empty matrix
```

```
waffle_chart = np.zeros((height, width), dtype = np.uint)

# 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
        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!
```

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

In [13]:

```
waffle_chart
```

Out[13]:

```
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]],
      dtype=uint64)
```

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.

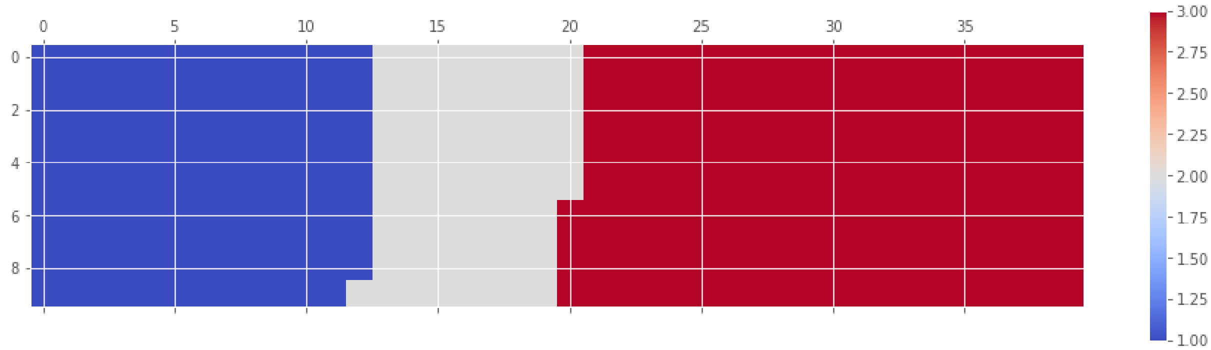
Step 5. Map the `waffle` chart matrix into a visual.

In [14]:

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

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

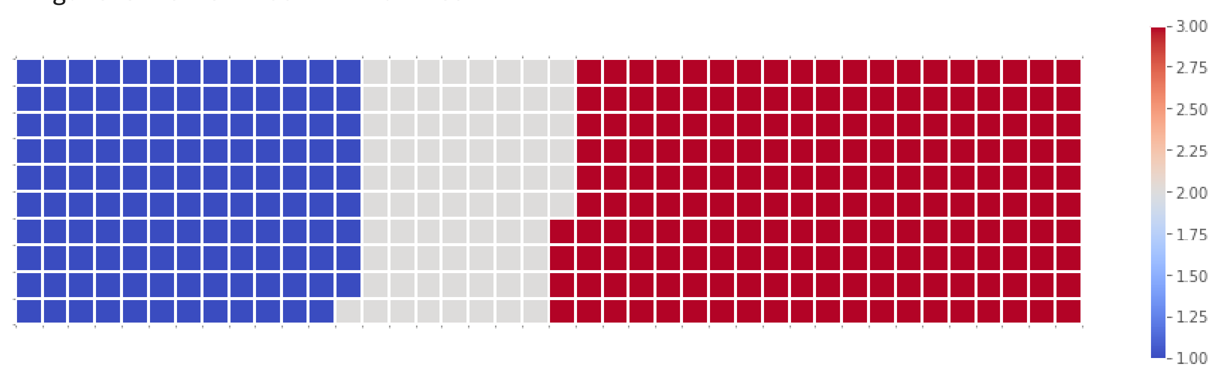
```
<Figure size 432x288 with 0 Axes>
```



Step 6. Prettify the chart.

In [15]:
```python
# 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([])
plt.show()
```

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

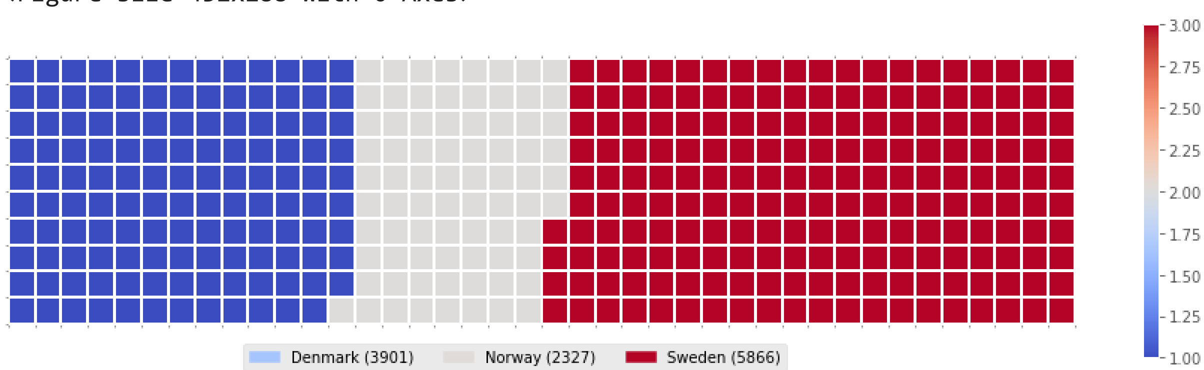


Step 7. Create a legend and add it to chart.

In [16]:

```
# 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([])

# compute cumulative sum of individual categories to match color schemes between cha
values_cumsum = np.cumsum(df_dsn['Total'])
total_values = values_cumsum[len(values_cumsum) - 1]

# create legend
legend_handles = []
for i, category in enumerate(df_dsn.index.values):
    label_str = category + ' (' + str(df_dsn['Total'][i]) + ')'
    color_val = colormap(float(values_cumsum[i])/total_values)
    legend_handles.append(mpatches.Patch(color=color_val, label=label_str))

# add legend to chart
plt.legend(handles=legend_handles,
           loc='lower center',
           ncol=len(df_dsn.index.values),
           bbox_to_anchor=(0., -0.2, 0.95, .1)
          )
plt.show()
```

<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 [17]:

```
def create_waffle_chart(categories, values, height, width, colormap, value_sign=''):

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

    # compute the total number of tiles
    total_num_tiles = width * height # total number of tiles
    print ('Total number of tiles is', total_num_tiles)

    # compute the number of tiles for each catagory
    tiles_per_category = [round(proportion * total_num_tiles) for proportion in cate

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

    # 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

    # 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)
```

```python
    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 dridlines based on minor ticks
    ax.grid(which='minor', color='w', linestyle='-', linewidth=2)

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

    # compute cumulative sum of individual categories to match color schemes between
    values_cumsum = np.cumsum(values)
    total_values = values_cumsum[len(values_cumsum) - 1]

    # create legend
    legend_handles = []
    for i, category in enumerate(categories):
        if value_sign == '%':
            label_str = category + ' (' + str(values[i]) + value_sign + ')'
        else:
            label_str = category + ' (' + value_sign + str(values[i]) + ')'

        color_val = colormap(float(values_cumsum[i])/total_values)
        legend_handles.append(mpatches.Patch(color=color_val, label=label_str))

    # add legend to chart
    plt.legend(
        handles=legend_handles,
        loc='lower center',
        ncol=len(categories),
        bbox_to_anchor=(0., -0.2, 0.95, .1)
    )
    plt.show()
```

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 [18]:

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

categories = df_dsn.index.values # categories
values = df_dsn['Total'] # correponding values of categories

colormap = plt.cm.coolwarm # color map class
```

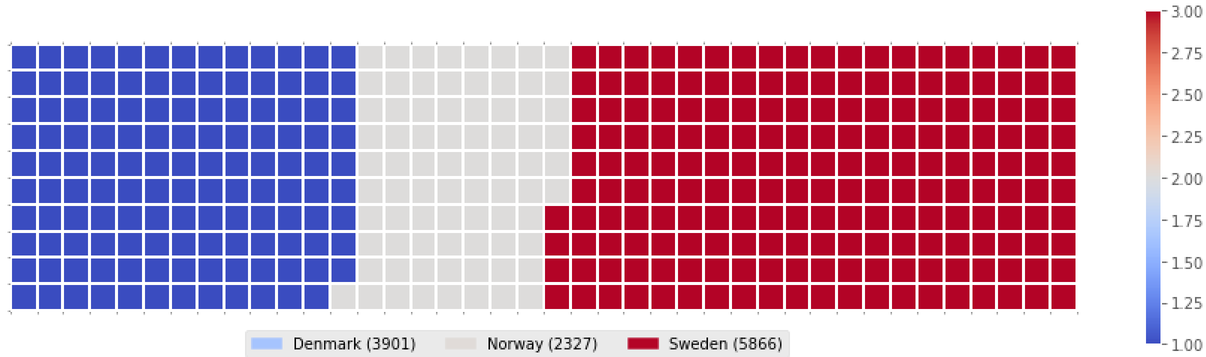
And now let's call our function to create a `waffle` chart.

In [19]:

```python
create_waffle_chart(categories, values, height, width, colormap)
```

```
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, 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.

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 [20]:

```
# install wordcloud
! pip install wordcloud

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

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

```
Collecting wordcloud
  Downloading https://files.pythonhosted.org/packages/05/e7/52e4bef8e2e3499f6e96cc8f
f7e0902a40b95014143b062acde4ff8b9fc8/wordcloud-1.8.1-cp36-cp36m-manylinux1_x86_64.wh
l (366kB)
     |████████████████████████████████| 368kB 29.2MB/s eta 0:00:01
Requirement already satisfied: pillow in /home/jupyterlab/conda/envs/python/lib/pyth
on3.6/site-packages (from wordcloud) (8.3.1)
Requirement already satisfied: matplotlib in /home/jupyterlab/conda/envs/python/lib/
python3.6/site-packages (from wordcloud) (3.3.4)
Requirement already satisfied: numpy>=1.6.1 in /home/jupyterlab/conda/envs/python/li
b/python3.6/site-packages (from wordcloud) (1.19.5)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /home/jup
```

```
yterlab/conda/envs/python/lib/python3.6/site-packages (from matplotlib->wordcloud)
(2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in /home/jupyterlab/conda/envs/p
ython/lib/python3.6/site-packages (from matplotlib->wordcloud) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/jupyterlab/conda/envs/pyth
on/lib/python3.6/site-packages (from matplotlib->wordcloud) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /home/jupyterlab/conda/envs/python/li
b/python3.6/site-packages/cycler-0.10.0-py3.6.egg (from matplotlib->wordcloud) (0.1
0.0)
Requirement already satisfied: six>=1.5 in /home/jupyterlab/conda/envs/python/lib/py
thon3.6/site-packages (from python-dateutil>=2.1->matplotlib->wordcloud) (1.15.0)
Installing collected packages: wordcloud
Successfully installed wordcloud-1.8.1
Wordcloud is installed and imported!
```

`Word` clouds are commonly used to perform high-level analysis and visualization of text data. Accordinly, 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 [21]:

```python
import urllib

# open the file and read it into a variable alice_novel
alice_novel = urllib.request.urlopen('https://cf-courses-data.s3.us.cloud-object-stc
```

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

In [22]:

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

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 [23]:

```python
# 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)
```

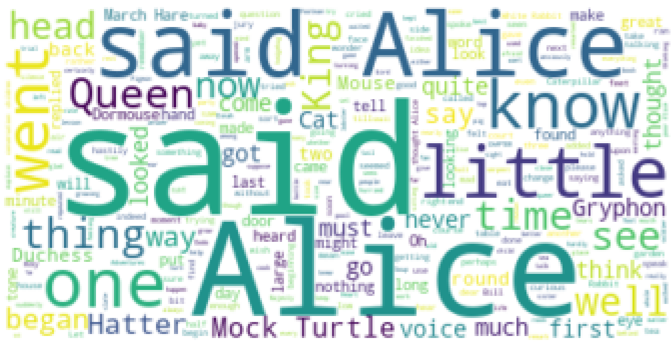
Out[23]:

```
<wordcloud.wordcloud.WordCloud at 0x7fbab54a8e10>
```

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

In [24]:

```python
# 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.

In [25]:
```
fig = plt.figure(figsize=(14, 18))

# 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 stopwords and re-generate the cloud.

In [26]:
```
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(figsize=(14, 18))

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

```
plt.axis('off')
plt.show()
```

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 [27]:
```
# save mask to alice_mask
alice_mask = np.array(Image.open(urllib.request.urlopen('https://cf-courses-data.s3.
```

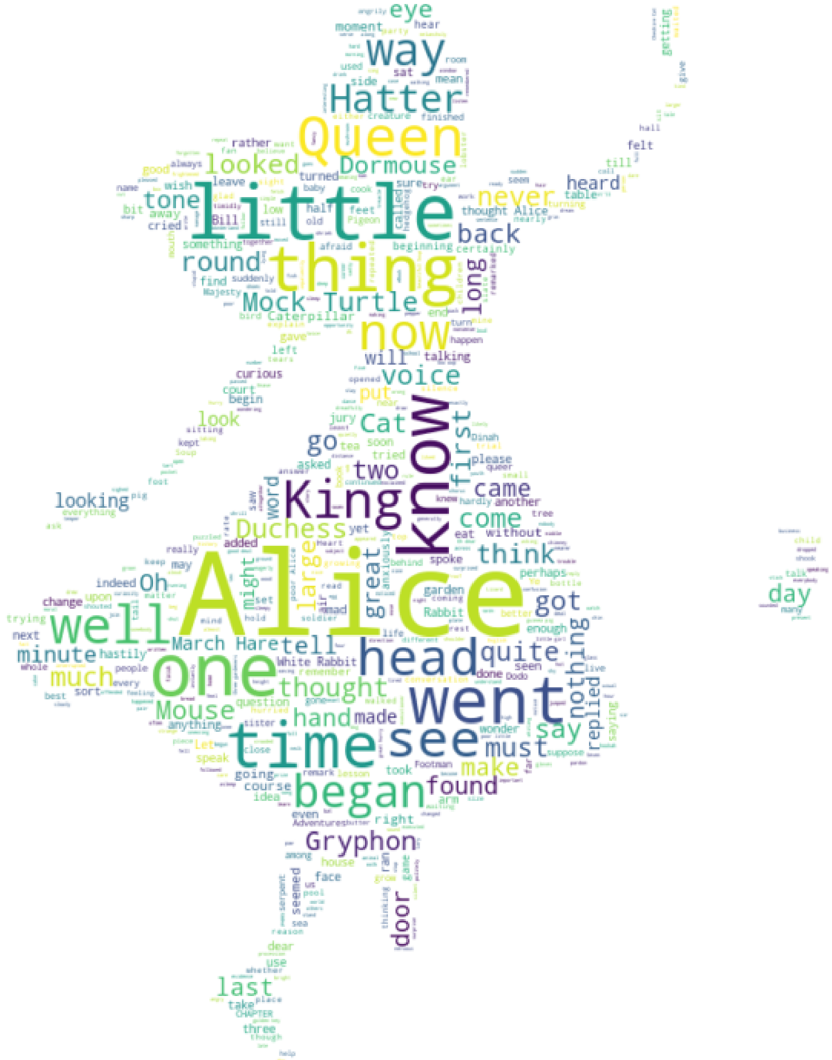
et's take a look at how the mask looks like.Let's take a look at how the mask looks like.

In [28]:
```
fig = plt.figure(figsize=(14, 18))

plt.imshow(alice_mask, cmap=plt.cm.gray, interpolation='bilinear')
plt.axis('off')
plt.show()
```

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 [29]:

```
# instantiate a word cloud object
alice_wc = WordCloud(background_color='white', max_words=2000, mask=alice_mask, stop

# generate the word cloud
alice_wc.generate(alice_novel)

# display the word cloud
fig = plt.figure(figsize=(14, 18))

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



Really impressive!

Unfortunately, our immigration 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 [30]:
```
df_can.head()
```

Out[30]:

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	200
Country												
Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	...	343

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	20(
Country												
Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	...	122
Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	...	362
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 [31]:

```
total_immigration = df_can['Total'].sum()
total_immigration
```

Out[31]:

```
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 [32]:

```
max_words = 90
word_string = ''
for country in df_can.index.values:
    # check if country's name is a single-word name
    if country.count(" ") == 0:
        repeat_num_times = int(df_can.loc[country, 'Total'] / total_immigration * ma
        word_string = word_string + ((country + ' ') * repeat_num_times)

# display the generated text
word_string
```

Out[32]:

```
'China China China China China China China China China Colombia Egypt France Guyana
Haiti India India India India India India India India Jamaica Lebanon Morocco
Pakistan Pakistan Pakistan Philippines Philippines Philippines Philippines Philippin
es Philippines Philippines Poland Portugal Romania '
```

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

In [33]:

```
# create the word cloud
wordcloud = WordCloud(background_color='white').generate(word_string)

print('Word cloud created!')
```

```
Word cloud created!
```

In [34]:
```
# display the cloud
plt.figure(figsize=(14, 18))

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

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 and more about *seaborn* regression plots by following this link.

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 [35]:
```
# install seaborn
! pip3 install seaborn

# import library
import seaborn as sns
```

```
print('Seaborn installed and imported!')
```

```
Collecting seaborn
  Downloading seaborn-0.11.2-py3-none-any.whl (292 kB)
     |████████████████████████████████| 292 kB 28.7 MB/s eta 0:00:01
Collecting pandas>=0.23
  Downloading pandas-1.3.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(11.5 MB)
     |████████████████████████████████| 11.5 MB 63.3 MB/s eta 0:00:01
Collecting numpy>=1.15
  Downloading numpy-1.21.2-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_64.whl
(15.8 MB)
     |████████████████████████████████| 15.8 MB 57.4 MB/s eta 0:00:01
Collecting matplotlib>=2.2
  Downloading matplotlib-3.4.3-cp38-cp38-manylinux1_x86_64.whl (10.3 MB)
     |████████████████████████████████| 10.3 MB 39.6 MB/s eta 0:00:01
Collecting scipy>=1.0
  Downloading scipy-1.7.1-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.whl (28.4
MB)
     |████████████████████████████████| 28.4 MB 60.0 MB/s eta 0:00:01
Collecting pytz>=2017.3
  Using cached pytz-2021.1-py2.py3-none-any.whl (510 kB)
Requirement already satisfied: python-dateutil>=2.7.3 in /home/jupyterlab/conda/lib/
python3.8/site-packages (from pandas>=0.23->seaborn) (2.8.1)
Collecting cycler>=0.10
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Requirement already satisfied: pyparsing>=2.2.1 in /home/jupyterlab/conda/lib/python
3.8/site-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.3.2-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.whl
(1.2 MB)
     |████████████████████████████████| 1.2 MB 74.8 MB/s eta 0:00:01
Collecting pillow>=6.2.0
  Downloading Pillow-8.3.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(3.0 MB)
     |████████████████████████████████| 3.0 MB 60.9 MB/s eta 0:00:01
Requirement already satisfied: six>=1.5 in /home/jupyterlab/conda/lib/python3.8/site
-packages (from python-dateutil>=2.7.3->pandas>=0.23->seaborn) (1.15.0)
Installing collected packages: pytz, numpy, pandas, cycler, kiwisolver, pillow, matp
lotlib, scipy, seaborn
Successfully installed cycler-0.10.0 kiwisolver-1.3.2 matplotlib-3.4.3 numpy-1.21.2
pandas-1.3.2 pillow-8.3.2 pytz-2021.1 scipy-1.7.1 seaborn-0.11.2
Seaborn installed and imported!
```

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

In [36]:

```
# 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 float (useful for regression later on)
df_tot.index = map(float, 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()
```

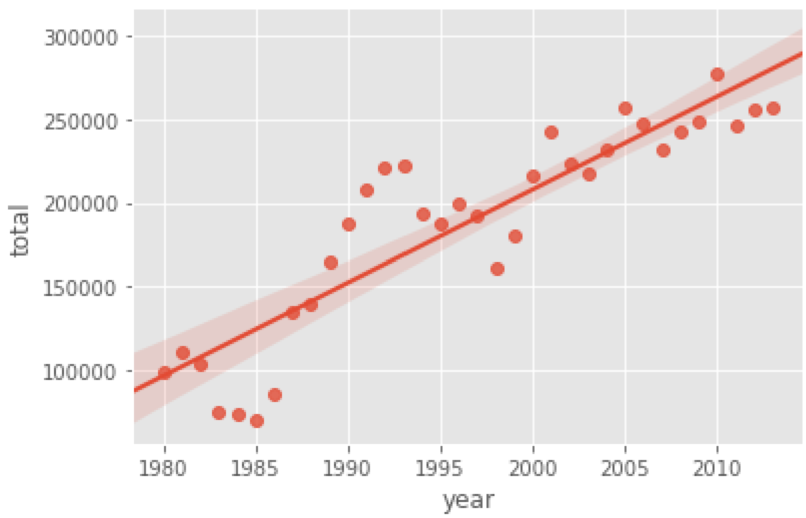
Out[36]:

	year	total
0	1980.0	99137
1	1981.0	110563
2	1982.0	104271
3	1983.0	75550
4	1984.0	73417

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

In [38]:

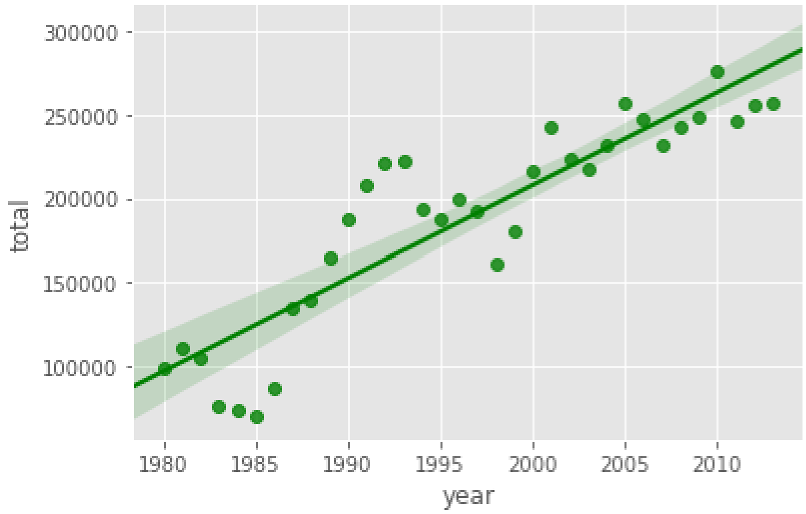
```
sns.regplot(x='year', y='total', data=df_tot)
plt.show()
```



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.

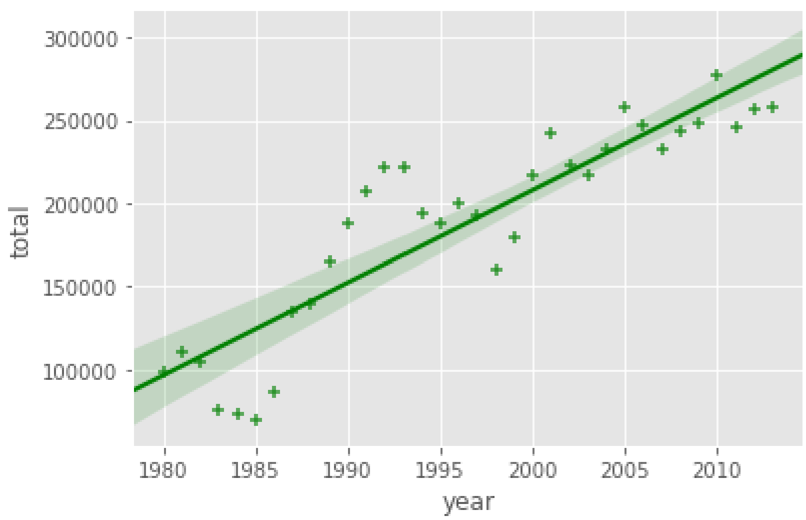
In [39]:

```
sns.regplot(x='year', y='total', data=df_tot, color='green')
plt.show()
```



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

In [40]:
```
ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+')
plt.show()
```



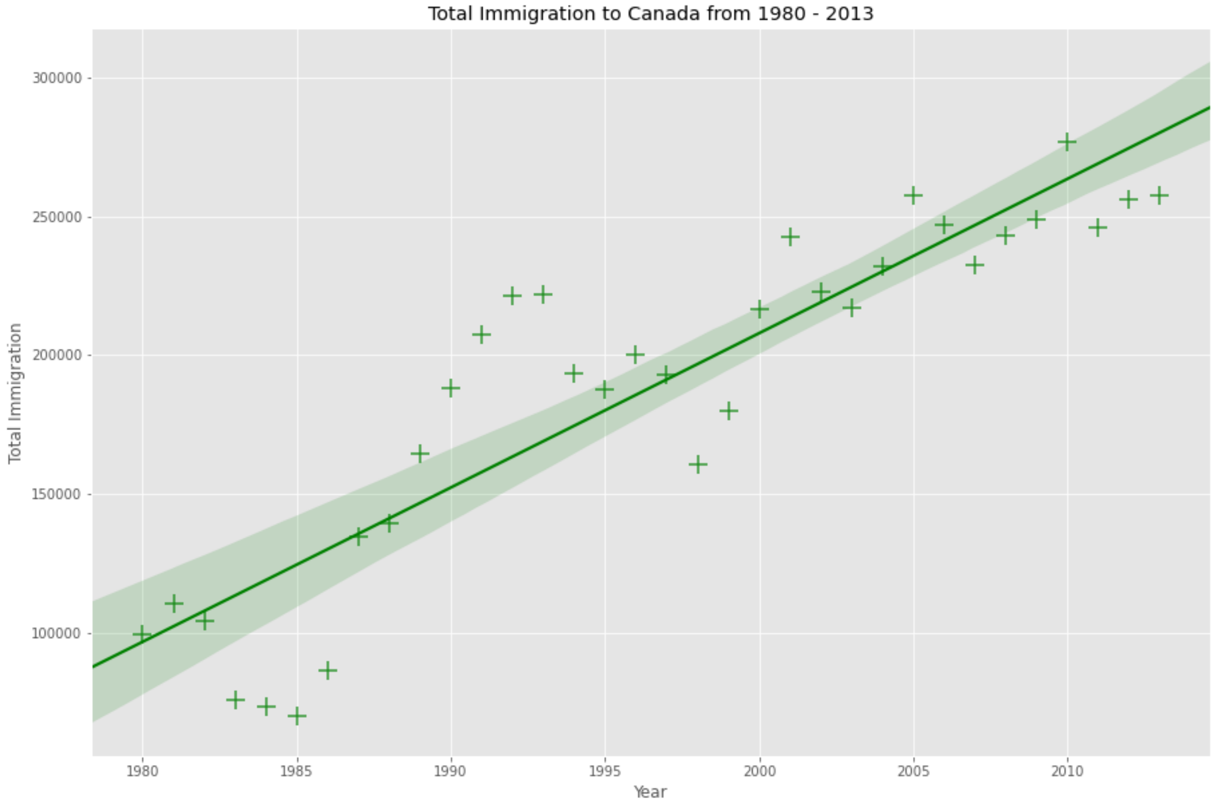
Let's blow up the plot a little so that it is more appealing to the sight.

In [42]:
```
plt.figure(figsize=(15, 10))
sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+')
plt.show()
```

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 [43]:

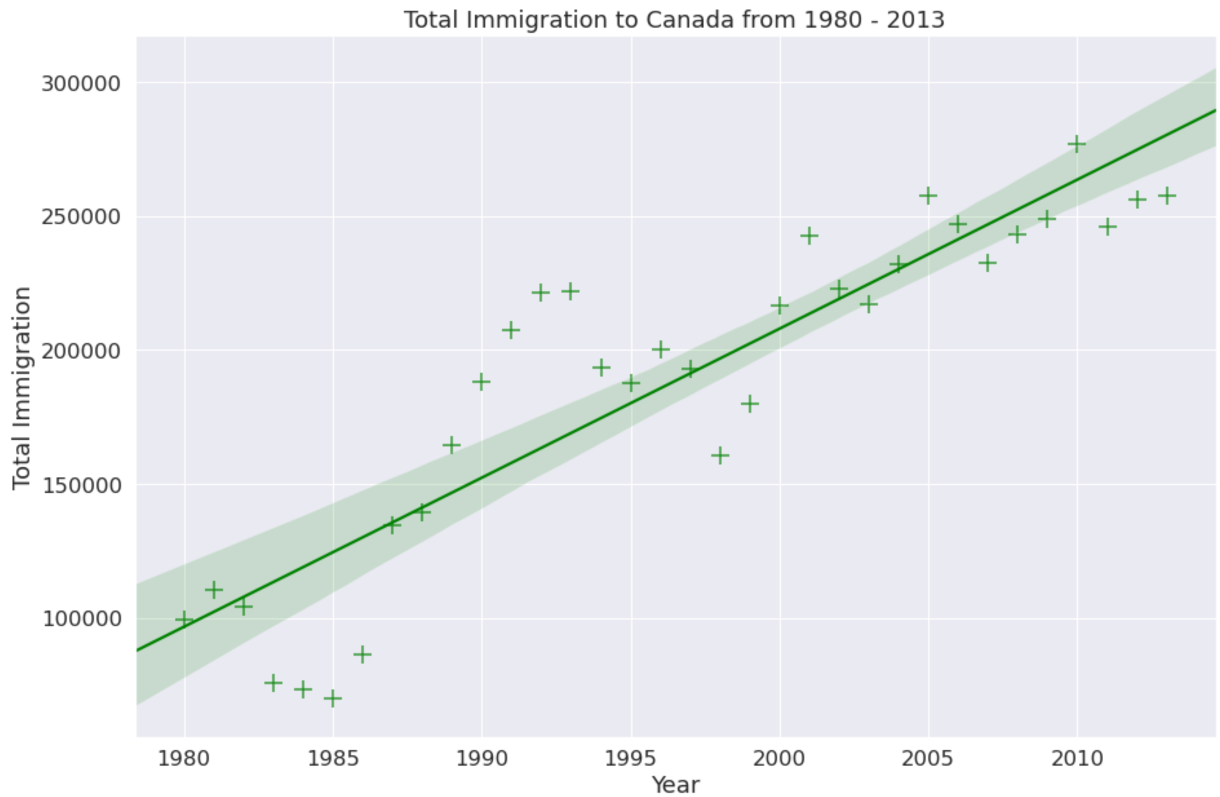
```
plt.figure(figsize=(15, 10))
ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+', scatte

ax.set(xlabel='Year', ylabel='Total Immigration') # add x- and y-labels
ax.set_title('Total Immigration to Canada from 1980 - 2013') # add title
plt.show()
```



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 [44]:

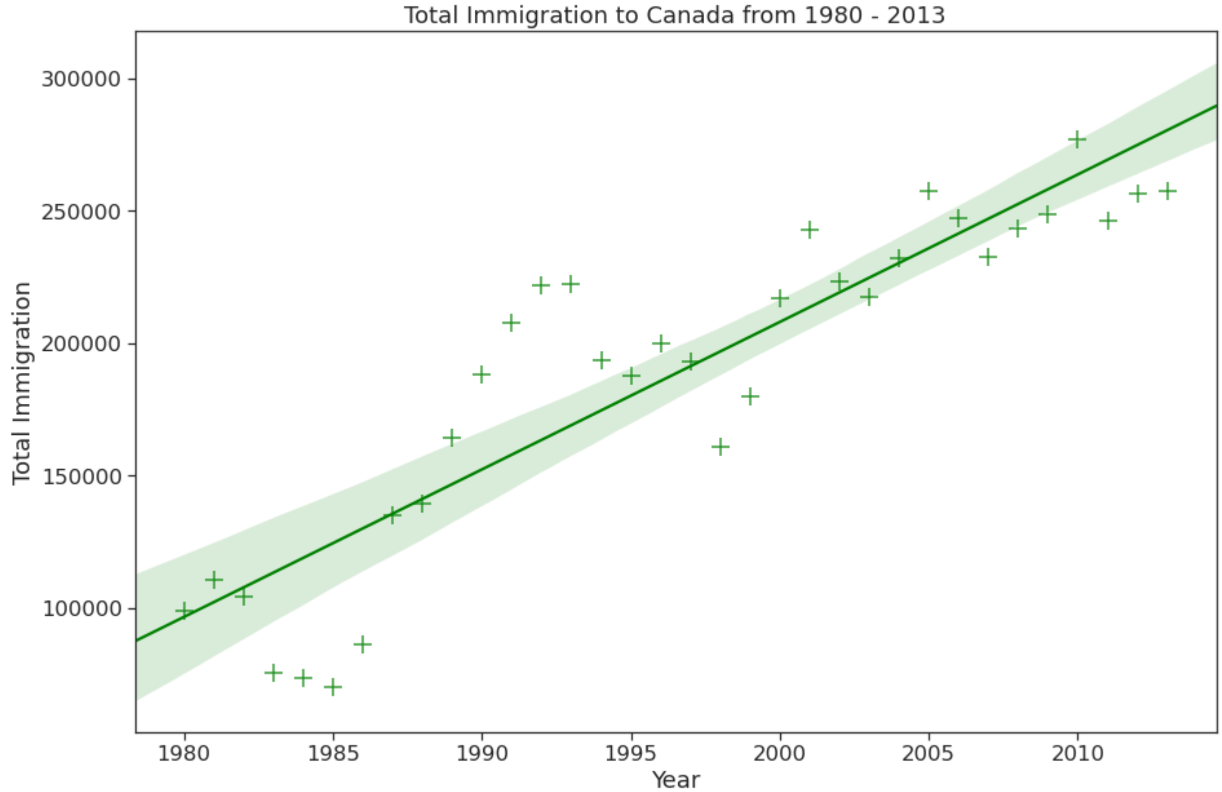
```
plt.figure(figsize=(15, 10))

sns.set(font_scale=1.5)

ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+', scatte
ax.set(xlabel='Year', ylabel='Total Immigration')
ax.set_title('Total Immigration to Canada from 1980 - 2013')
plt.show()
```



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.

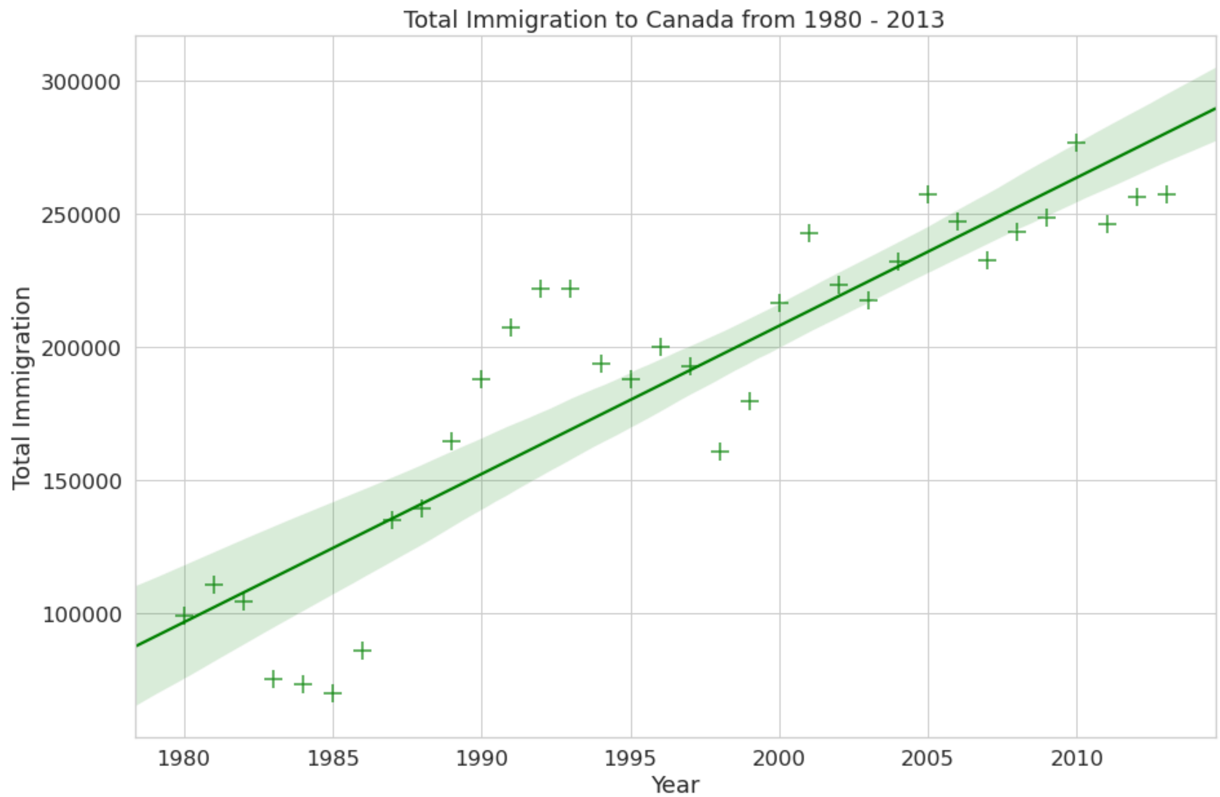
In [45]:
```
plt.figure(figsize=(15, 10))

sns.set(font_scale=1.5)
sns.set_style('ticks')  # change background to white background

ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+', scatte
ax.set(xlabel='Year', ylabel='Total Immigration')
ax.set_title('Total Immigration to Canada from 1980 - 2013')
plt.show()
```



Or to a white background with gridlines.

In [46]:

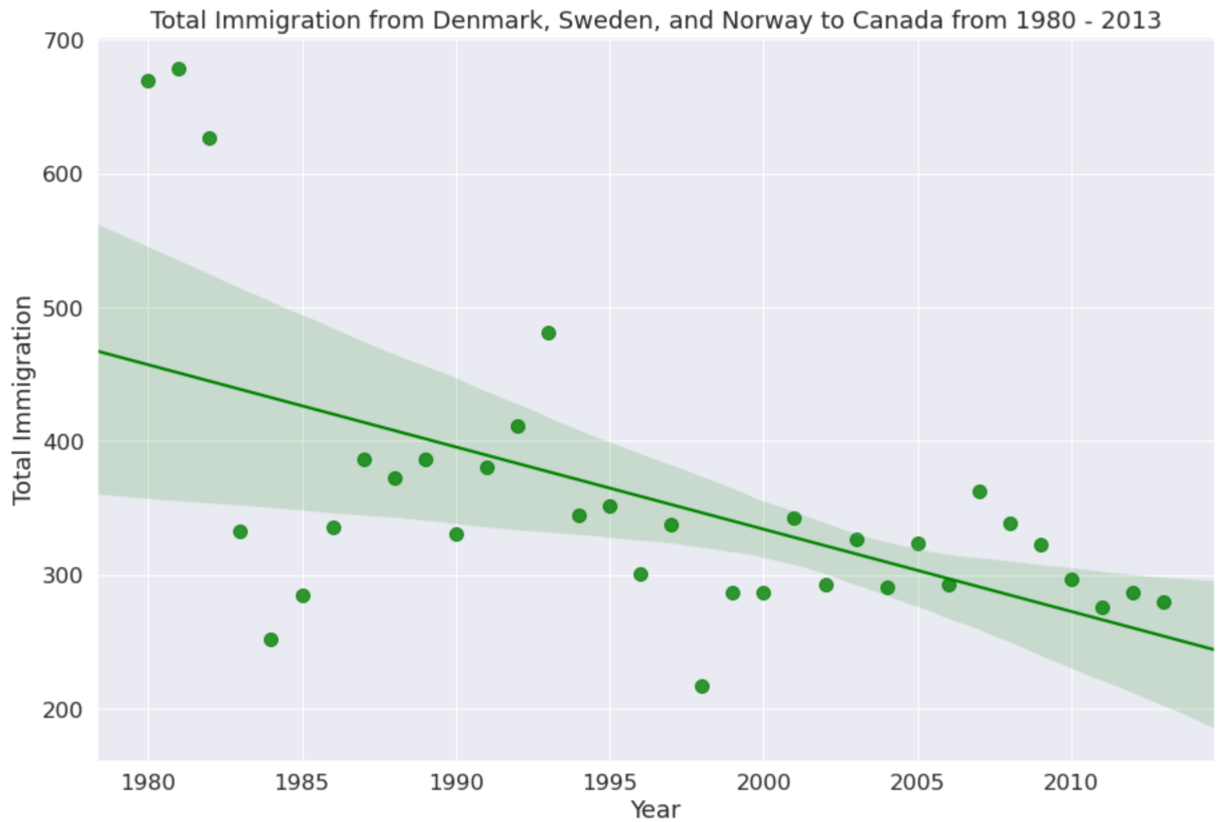
```
plt.figure(figsize=(15, 10))

sns.set(font_scale=1.5)
sns.set_style('whitegrid')

ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+', scatte
ax.set(xlabel='Year', ylabel='Total Immigration')
ax.set_title('Total Immigration to Canada from 1980 - 2013')
plt.show()
```



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 [47]:

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

df_total = pd.DataFrame(df_countries.sum(axis=1))

df_total.reset_index(inplace=True)

df_total.columns = ['year', 'total']

df_total['year'] = df_total['year'].astype(int)

plt.figure(figsize=(15, 10))

sns.set(font_scale=1.5)

ax = sns.regplot(x='year', y='total', data=df_total, color='green', scatter_kws={'s'
ax.set(xlabel='Year', ylabel='Total Immigration')
ax.set_title('Total Immigration from Denmark, Sweden, and Norway to Canada from 1980
plt.show()
```



Click here for a sample python solution ```python #The correct answer is: # create df_countries dataframe df_countries = df_can.loc[['Denmark', 'Norway', 'Sweden'], years].transpose() # create df_total by summing across three countries for each year df_total = pd.DataFrame(df_countries.sum(axis=1)) # reset index in place df_total.reset_index(inplace=True) # rename columns df_total.columns = ['year', 'total'] # change column year from string to int to create scatter plot df_total['year'] = df_total['year'].astype(int) # define figure size plt.figure(figsize=(15, 10)) # define background style and font size sns.set(font_scale=1.5) sns.set_style('whitegrid') # generate plot and add title and axes labels ax = sns.regplot(x='year', y='total', data=df_total, color='green', marker='+', scatter_kws={'s': 200}) ax.set(xlabel='Year', ylabel='Total Immigration') ax.set_title('Total Immigrationn from Denmark, Sweden, and Norway to Canada from 1980 - 2013') ```

Thank you for completing this lab!

Author

Alex Aklson

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-05-19	2.3	Weiqing Wang	Fixed typos and code smells

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-01-21	2.2	Lakshmi Holla	Updated TOC markdown cell
2020-11-03	2.1	Lakshmi Holla	Changed URL of excel file
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab