DV0101EN-Exercise-Waffle-Charts-Word-Clouds-and-Regression-Plots

September 26, 2025

1 Waffle Charts, Word Clouds, and Regression Plots

Estimated time needed: **40** minutes

1.1 Objectives

After completing this lab you will be able to:

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

1.2 Table of Contents

2 Import Libraries

[1]:
```
!pip install matplotlib
!pip install pandas
```

```
Collecting matplotlib
  Downloading matplotlib-3.10.6-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.3-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl.metadata (111 kB)
```

```
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.9-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (6.3 kB)
Collecting numpy>=1.23 (from matplotlib)
  Downloading
numpy-2.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata
(62 kB)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib) (24.2)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.3.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (9.0 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Downloading pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading
matplotlib-3.10.6-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl
(8.7 MB)
                         8.7/8.7 MB
142.5 MB/s eta 0:00:00
Downloading
contourpy-1.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (362
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl (4.9 MB)
                         4.9/4.9 MB
98.7 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (1.5
MB)
                         1.5/1.5 MB
92.1 MB/s eta 0:00:00
Downloading
numpy-2.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.6
MB)
                         16.6/16.6 MB
162.6 MB/s eta 0:00:00
Downloading
pillow-11.3.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (6.6
MB)
                         6.6/6.6 MB
150.8 MB/s eta 0:00:00
Downloading pyparsing-3.2.5-py3-none-any.whl (113 kB)
```

```
Installing collected packages: pyparsing, pillow, numpy, kiwisolver, fonttools,
cycler, contourpy, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.60.0
kiwisolver-1.4.9 matplotlib-3.10.6 numpy-2.3.3 pillow-11.3.0 pyparsing-3.2.5
Collecting pandas
  Downloading
pandas-2.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(91 kB)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2.3.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2024.2)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading
pandas-2.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.0
MB)
                         12.0/12.0 MB
112.0 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, pandas
Successfully installed pandas-2.3.2 tzdata-2025.2
```

[2]:

```
#Import and setup matplotlib:
%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

#Import Primary Modules:
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

#install seaborn and wordcloud
!pip install seaborn wordcloud

#import seaborn
import seaborn as sns
```

```python
#import wordcloud
import wordcloud

# check for latest version of Matplotlib and seaborn
print ('Matplotlib version: ', mpl.__version__) # >= 2.0.0
print('Seaborn version: ', sns.__version__)
print('WordCloud version: ', wordcloud.__version__)
```

```
Collecting seaborn
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Collecting wordcloud
  Downloading wordcloud-1.9.4-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/opt/conda/lib/python3.12/site-packages (from seaborn) (2.3.3)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-
packages (from seaborn) (2.3.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/opt/conda/lib/python3.12/site-packages (from seaborn) (3.10.6)
Requirement already satisfied: pillow in /opt/conda/lib/python3.12/site-packages
(from wordcloud) (11.3.0)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(4.60.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(1.4.9)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(24.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
```

```
Downloading
wordcloud-1.9.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (539
kB)
                         539.2/539.2 kB
27.9 MB/s eta 0:00:00
Installing collected packages: wordcloud, seaborn
Successfully installed seaborn-0.13.2 wordcloud-1.9.4
Matplotlib version:  3.10.6
Seaborn version:  0.13.2
WordCloud version:  1.9.4
```

3 Fetching Data

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 and use the *already cleaned dataset*. You can refer to the lab on data pre-processing wherein this dataset is cleaned for a quick refresh your Panads skill Data pre-processing with Pandas

Download the Canadian Immigration dataset and read it into a *pandas* dataframe.

[3]:
```
df_can = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.
↪appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/
↪Data%20Files/Canada.csv')

print('Data read into a pandas dataframe!')
```

```
Data read into a pandas dataframe!
```

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

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

[4]:
```
          Country Continent           Region             DevName  1980  1981  \
0     Afghanistan      Asia    Southern Asia  Developing regions    16    39
1         Albania    Europe  Southern Europe   Developed regions     1     0
2         Algeria    Africa  Northern Africa  Developing regions    80    67
3  American Samoa   Oceania        Polynesia  Developing regions     0     1
4         Andorra    Europe  Southern Europe   Developed regions     0     0

   1982  1983  1984  1985  …  2005  2006  2007  2008  2009  2010  2011  \
0    39    47    71   340  …  3436  3009  2652  2111  1746  1758  2203
1     0     0     0     0  …  1223   856   702   560   716   561   539
```

```
2     71     69     63     44  ...  3626  4807  3623  4005  5393  4752  4325
3      0      0      0      0  ...     0     1     0     0     0     0     0
4      0      0      0      0  ...     0     1     1     0     0     0     0

   2012  2013  Total
0  2635  2004  58639
1   620   603  15699
2  3774  4331  69439
3     0     0      6
4     1     1     15

[5 rows x 39 columns]
```

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

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

```
(195, 39)
```

[6]:
```python
#set Country as index
df_can.set_index('Country', inplace=True)
```

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.

[7]:
```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
```

[7]:
```
        Continent           Region             DevName  1980  1981  1982  1983  \
Country
Denmark    Europe  Northern Europe  Developed regions   272   293   299   106
Norway     Europe  Northern Europe  Developed regions   116    77   106    51
Sweden     Europe  Northern Europe  Developed regions   281   308   222   176

         1984  1985  1986  ...  2005  2006  2007  2008  2009  2010  2011  \
Country                    ...
Denmark    93    73    93  ...    62   101    97   108    81    92    93
Norway     31    54    56  ...    57    53    73    66    75    46    49
Sweden    128   158   187  ...   205   139   193   165   167   159   134
```

```
         2012  2013  Total
Country
Denmark    94    81   3901
Norway     53    59   2327
Sweden    140   140   5866

[3 rows x 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.

[8]:
```python
# 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})
```

[8]:
```
         Category Proportion
Country
Denmark             0.322557
Norway              0.192409
Sweden              0.485034
```

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

[9]:
```python
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

[10]:
```python
# 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})
```

[10]:
```
         Number of tiles
Country
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.

[11]:
```python
# 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 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!
```

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

[12]:
```python
waffle_chart
```

[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,
```

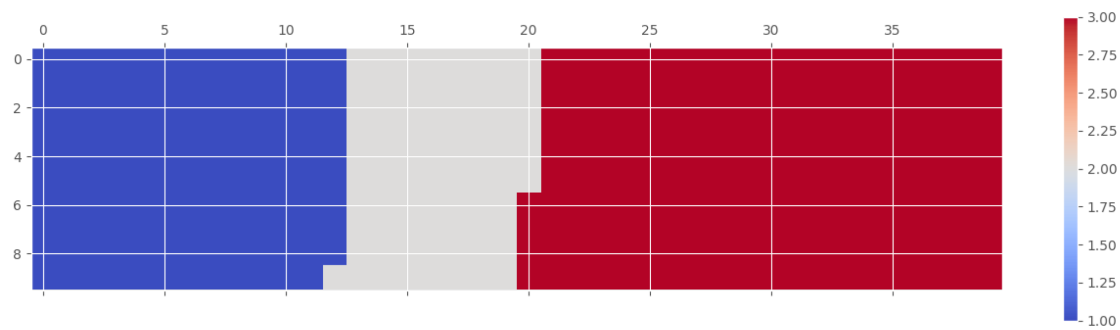
```
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 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]],
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.

[13]:
```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()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

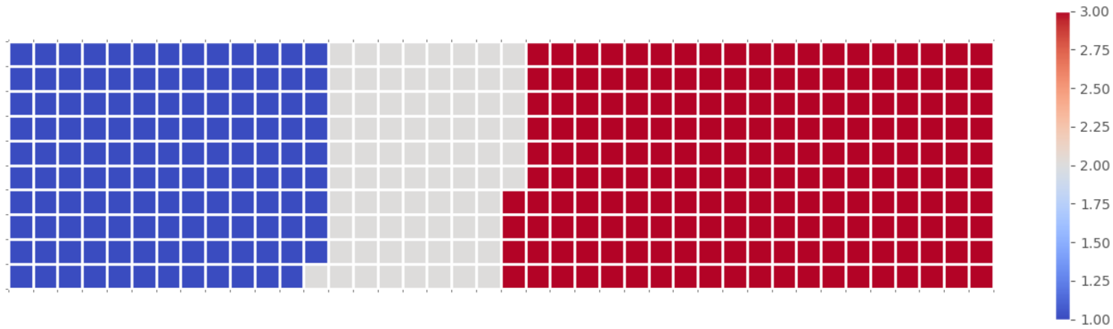


Step 6. Prettify the chart.

[14]:
```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 640x480 with 0 Axes>
```



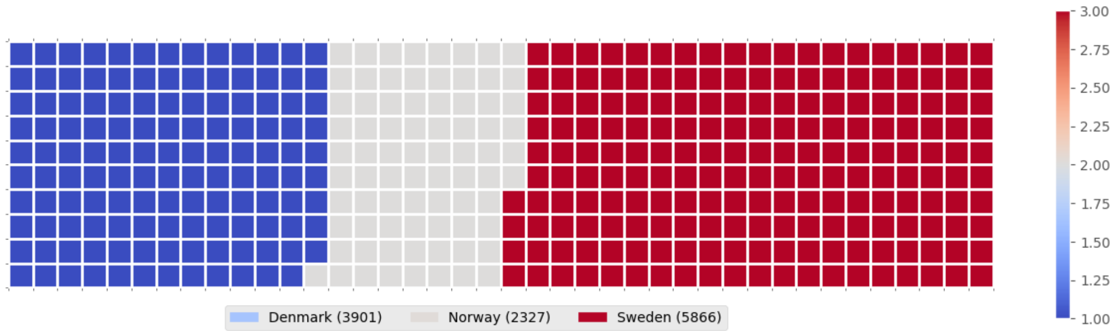
Step 7. Create a legend and add it to chart.

[15]:
```
# 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)
```

```python
# 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 chart and legend
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()
```

```
/tmp/ipykernel_299/2463873726.py:24: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  total_values = values_cumsum[len(values_cumsum) - 1]
/tmp/ipykernel_299/2463873726.py:29: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  label_str = category + ' (' + str(df_dsn['Total'][i]) + ')'
/tmp/ipykernel_299/2463873726.py:30: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  color_val = colormap(float(values_cumsum[i])/total_values)
```

```
<Figure size 640x480 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.

[16]:
```python
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 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))

    # 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)
    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 chart and legend
    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:

[17]:
```
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.

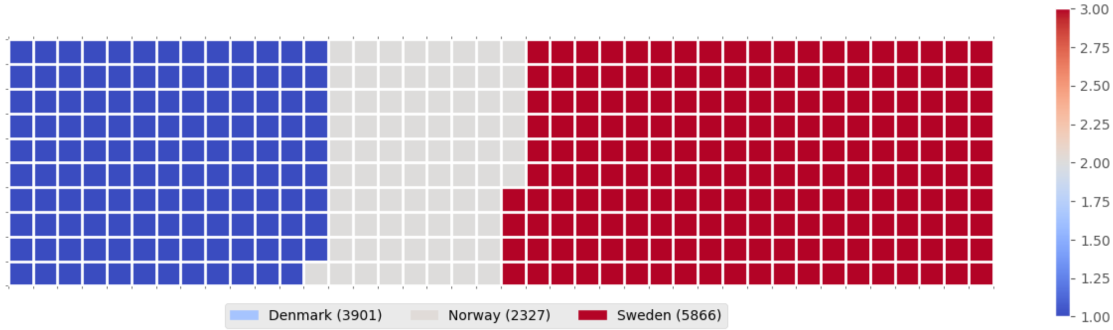
[18]:
```
create_waffle_chart(categories, values, height, width, colormap)
```

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

```
/tmp/ipykernel_299/3286913405.py:62: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  total_values = values_cumsum[len(values_cumsum) - 1]
/tmp/ipykernel_299/3286913405.py:70: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  label_str = category + ' (' + value_sign + str(values[i]) + ')'
```

```
/tmp/ipykernel_299/3286913405.py:72: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  color_val = colormap(float(values_cumsum[i])/total_values)
```

```
<Figure size 640x480 with 0 Axes>
```



There seems to be a new Python package for generating **waffle charts** called PyWaffle, Let's create the same waffle chart with **pywaffle** now

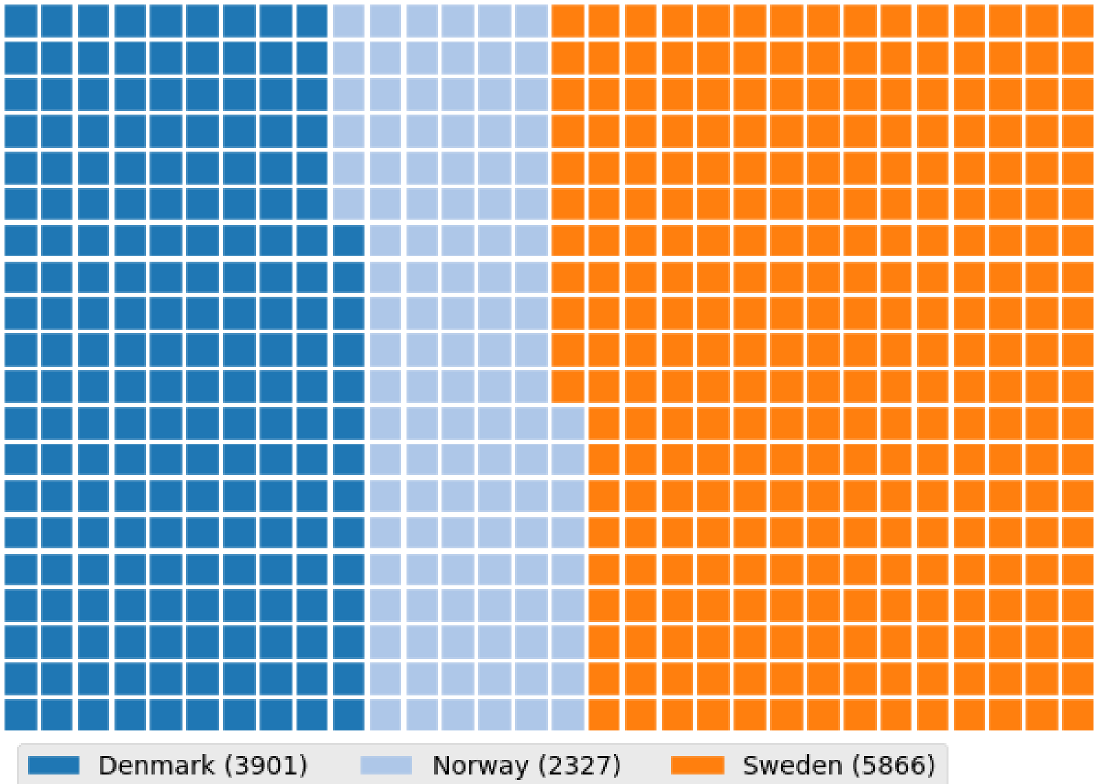
[19]:
```
#install pywaffle
!pip install pywaffle
```

```
Collecting pywaffle
  Downloading pywaffle-1.1.1-py2.py3-none-any.whl.metadata (2.6 kB)
Collecting fontawesomefree (from pywaffle)
  Downloading fontawesomefree-6.6.0-py3-none-any.whl.metadata (853 bytes)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.12/site-
packages (from pywaffle) (3.10.6)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib->pywaffle) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-
packages (from matplotlib->pywaffle) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib->pywaffle) (4.60.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib->pywaffle) (1.4.9)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib->pywaffle) (2.3.3)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib->pywaffle) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-
packages (from matplotlib->pywaffle) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib->pywaffle) (3.2.5)
```

```
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib->pywaffle)
(2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib->pywaffle) (1.17.0)
Downloading pywaffle-1.1.1-py2.py3-none-any.whl (30 kB)
Downloading fontawesomefree-6.6.0-py3-none-any.whl (25.6 MB)
                          25.6/25.6 MB
122.9 MB/s eta 0:00:0000:01
Installing collected packages: fontawesomefree, pywaffle
Successfully installed fontawesomefree-6.6.0 pywaffle-1.1.1
```

[20]:

```python
#import Waffle from pywaffle
from pywaffle import Waffle

#Set up the Waffle chart figure

fig = plt.figure(FigureClass = Waffle,
                 rows = 20, columns = 30, #pass the number of rows and columns␣
 ↪for the waffle
                 values = df_dsn['Total'], #pass the data to be used for display
                 cmap_name = 'tab20', #color scheme
                 legend = {'labels': [f"{k} ({v})" for k, v in zip(df_dsn.index.
 ↪values,df_dsn.Total)],
                           'loc': 'lower left', 'bbox_to_anchor':(0,-0.
 ↪1),'ncol': 3}
                 #notice the use of list comprehension for creating labels
                 #from index and total of the dataset
                )

#Display the waffle chart
plt.show()
```



Question: Create a Waffle chart to dispaly the proportiona of China and Inida total immigrant contribution.

[ ]:

Click here for a sample python solution

```
#hint
#create dataframe
data_CI = .............
#Set up the Waffle chart figure

fig = plt.figure(FigureClass = ............,
                 rows = ........, columns =....., #pass the number of rows and columns for
                 values = ........., #pass the data to be used for display
                 cmap_name = 'tab20', #color scheme
                 legend = {'labels':[.......],
                           'loc': ........., 'bbox_to_anchor':(....),'ncol': 2}
                 #notice the use of list comprehension for creating labels
                 #from index and total of the dataset
                )

#Display the waffle chart
```

```
plt.show()
```

5 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.

[21]:
```
#import package and its set of stopwords
from wordcloud import WordCloud, STOPWORDS

print ('Wordcloud imported!')
```

```
Wordcloud 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.

[22]:
```
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-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/alice_novel.
↪txt').read().decode("utf-8")
```

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

[23]:
```
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.

[24]:
```
#if you get attribute error while generating worldcloud, upgrade Pillow and␣
↪numpy using below code
#%pip install --upgrade Pillow
#%pip install --upgrade numpy
```

[25]:
```
# instantiate a word cloud object
alice_wc = WordCloud()
```

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

[25]:

```
<wordcloud.wordcloud.WordCloud at 0x7a2bbb03cb30>
```

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

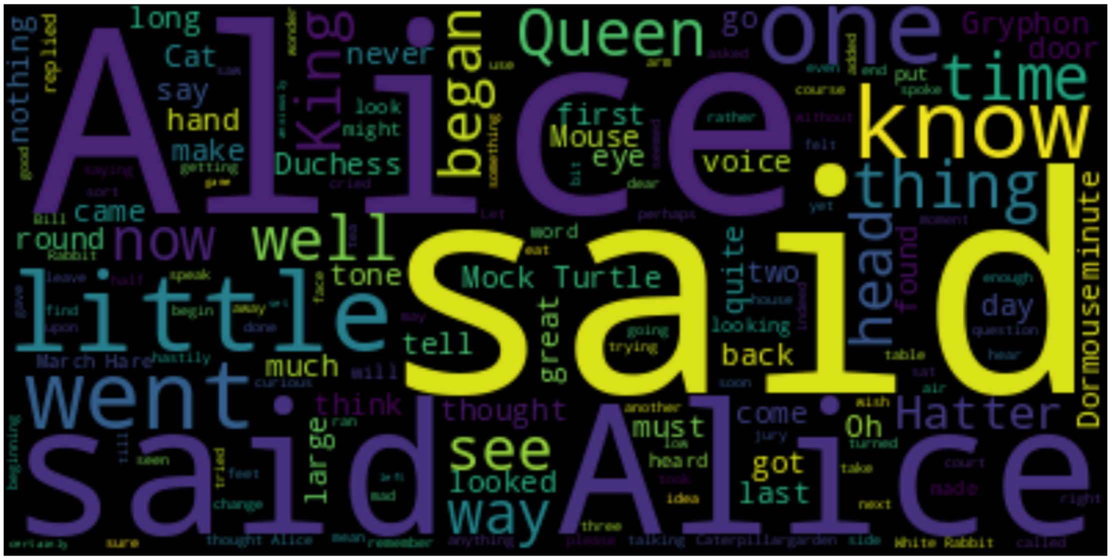
[26]:

```
# 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.

[27]:

```
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.

[28]:
```
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*.

[29]:
```python
#save mask to alice_mask
alice_mask = np.array(Image.open(urllib.request.urlopen('https://
 ↪cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/labs/Module%204/images/
 ↪alice_mask.png')))
```

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

[30]:
```python
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.

[31]:
```
# instantiate a word cloud object
alice_wc = WordCloud(background_color='white', max_words=2000, mask=alice_mask,␣
↪stopwords=stopwords)

# 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.

[32]:
```
df_can.head()
```

[32]:

```
                    Continent           Region             DevName  1980  1981  \
Country
Afghanistan              Asia    Southern Asia  Developing regions    16    39
Albania                Europe  Southern Europe   Developed regions     1     0
Algeria                Africa  Northern Africa  Developing regions    80    67
American Samoa        Oceania        Polynesia  Developing regions     0     1
Andorra                Europe  Southern Europe   Developed regions     0     0

                1982  1983  1984  1985  1986  …  2005  2006  2007  2008  \
Country                                       …
Afghanistan       39    47    71   340   496  …  3436  3009  2652  2111
Albania            0     0     0     0     1  …  1223   856   702   560
Algeria           71    69    63    44    69  …  3626  4807  3623  4005
American Samoa     0     0     0     0     0  …     0     1     0     0
Andorra            0     0     0     0     2  …     0     1     1     0

                2009  2010  2011  2012  2013  Total
Country
Afghanistan     1746  1758  2203  2635  2004  58639
Albania          716   561   539   620   603  15699
Algeria         5393  4752  4325  3774  4331  69439
American Samoa     0     0     0     0     0      6
Andorra            0     0     0     1     1     15

[5 rows x 38 columns]
```

And what was the total immigration from 1980 to 2013?

[33]:

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

[33]:

```
np.int64(6409153)
```

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

[34]:

```python
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␣
 ↪* max_words)
        word_string = word_string + ((country + ' ') * repeat_num_times)

# display the generated text
word_string
```

[34]: 'China China China China China China China China China Colombia Egypt France
Guyana Haiti India India India India India India India India India Jamaica
Lebanon Morocco Pakistan Pakistan Pakistan Philippines Philippines Philippines
Philippines Philippines 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.

[35]:
```
# create the word cloud
wordcloud = WordCloud(background_color='white').generate(word_string)

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

Word cloud created!

[36]:
```
# 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!

6 Plotting with Seaborn

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!

6.0.1 Categorical Plots

In our data 'df_can', let's find out how many continents are mentioned

[37]:
```
df_can['Continent'].unique()
```

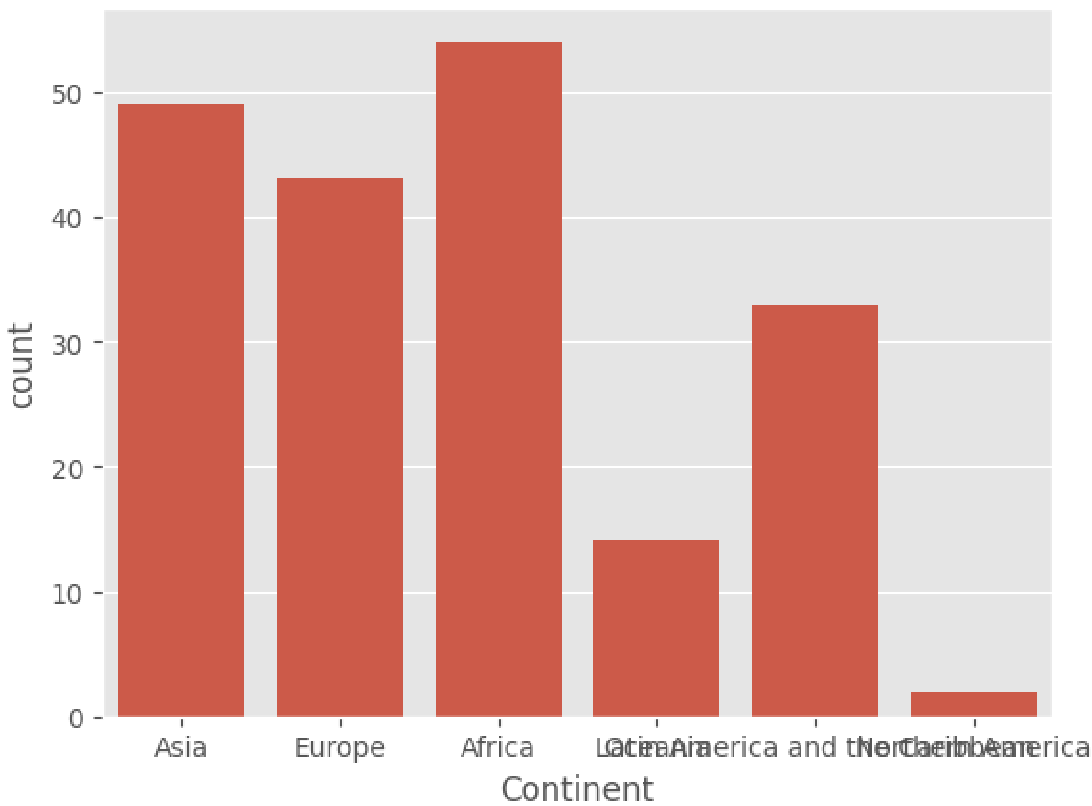
[37]:
```
array(['Asia', 'Europe', 'Africa', 'Oceania',
       'Latin America and the Caribbean', 'Northern America'],
      dtype=object)
```

6.0.2 countplot

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. Let's find the count of Continents in the data 'df_can' using countplot on 'Continent'

[38]:
```
sns.countplot(x='Continent', data=df_can)
```

[38]:
```
<Axes: xlabel='Continent', ylabel='count'>
```

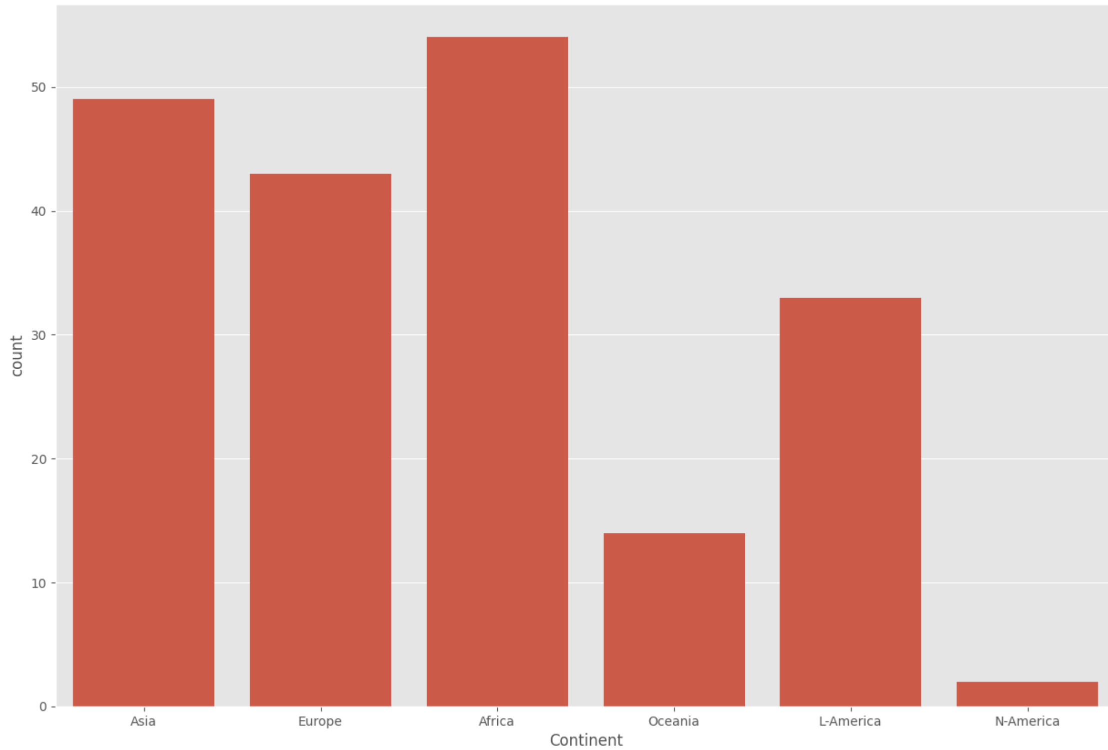


The labels on the x-axis doesnot look as expected.Let's try to replace the 'Latin America and the Caribbean' with and "L-America", 'Northern America' with "N-America", and change the figure size and then display the plot again

[39]:
```
df_can1 = df_can.replace('Latin America and the Caribbean', 'L-America')
df_can1 = df_can1.replace('Northern America', 'N-America')
```

[40]:
```
plt.figure(figsize=(15, 10))
sns.countplot(x='Continent', data=df_can1)
```

[40]:
```
<Axes: xlabel='Continent', ylabel='count'>
```



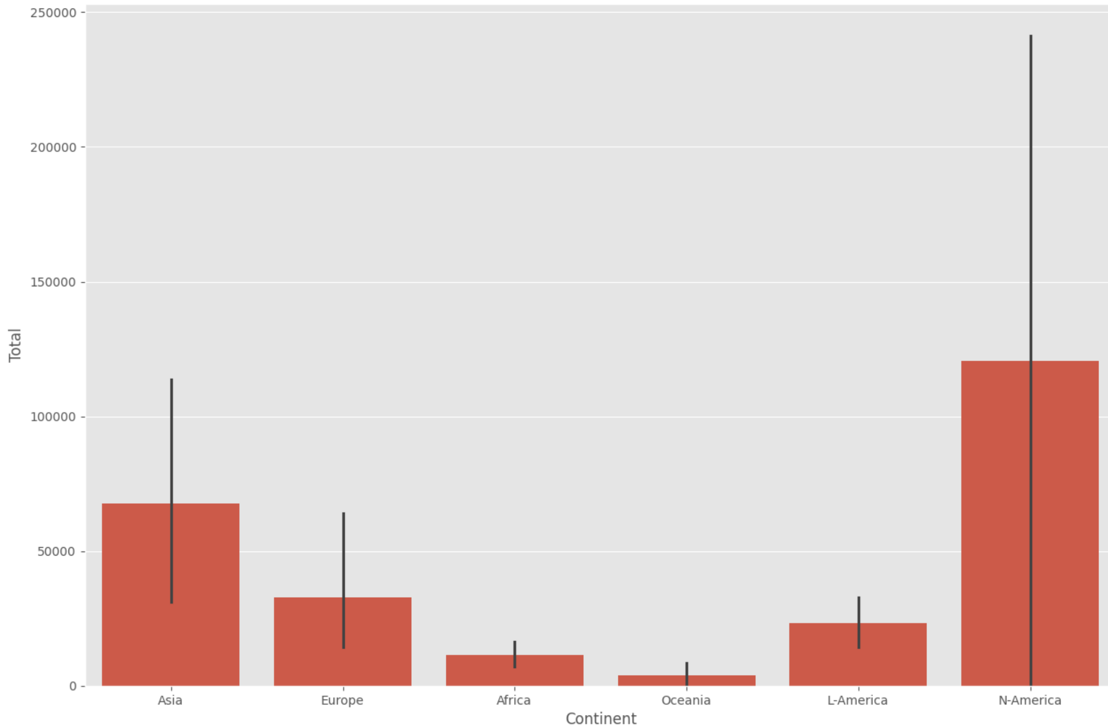
Much better!

6.0.3 Barplot

This plot will perform the Groupby on a categorical varaible and plot aggregated values, with confidence intervals. Let's plot the total immigrants Continent-wise

[41]:
```
plt.figure(figsize=(15, 10))
sns.barplot(x='Continent', y='Total', data=df_can1)
```

[41]: <Axes: xlabel='Continent', ylabel='Total'>



You can verify the values by performing the groupby on the Total and Continent for mean()

[42]:
```
df_Can2=df_can1.groupby('Continent')['Total'].mean()
df_Can2
```

[42]:
```
Continent
Africa         11462.000000
Asia           67710.081633
Europe         32812.720930
L-America      23186.303030
N-America     120571.000000
Oceania         3941.000000
Name: Total, dtype: float64
```

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

7 Regression Plot

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

[43]:
```
years = list(map(str, range(1980, 2014)))
# we can use the sum() method to get the total population per year
df_tot = pd.DataFrame(df_can[years].sum(axis=0))
```

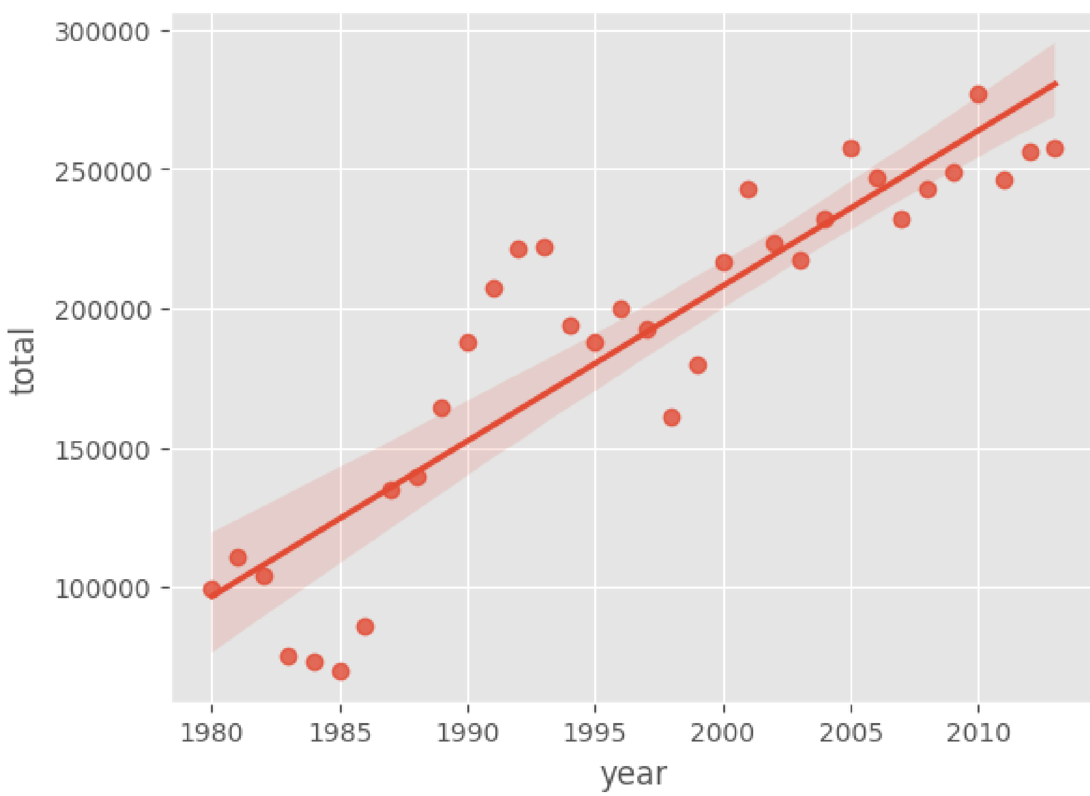
```python
# change the years to type float (useful for regression later on)
df_tot.index = map(float, df_tot.index)

# 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()
```

[43]:

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

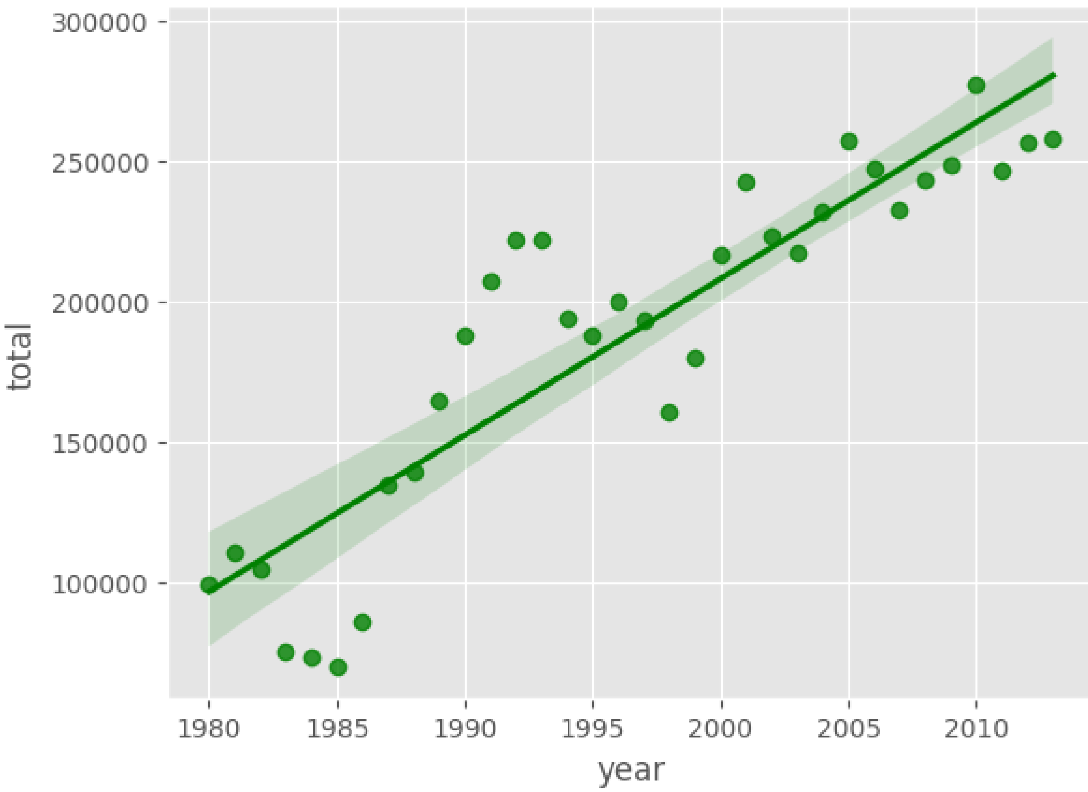
[44]:
```python
#seaborn is already imported at the start of this lab
sns.regplot(x='year', y='total', data=df_tot)
```

[44]: <Axes: xlabel='year', ylabel='total'>



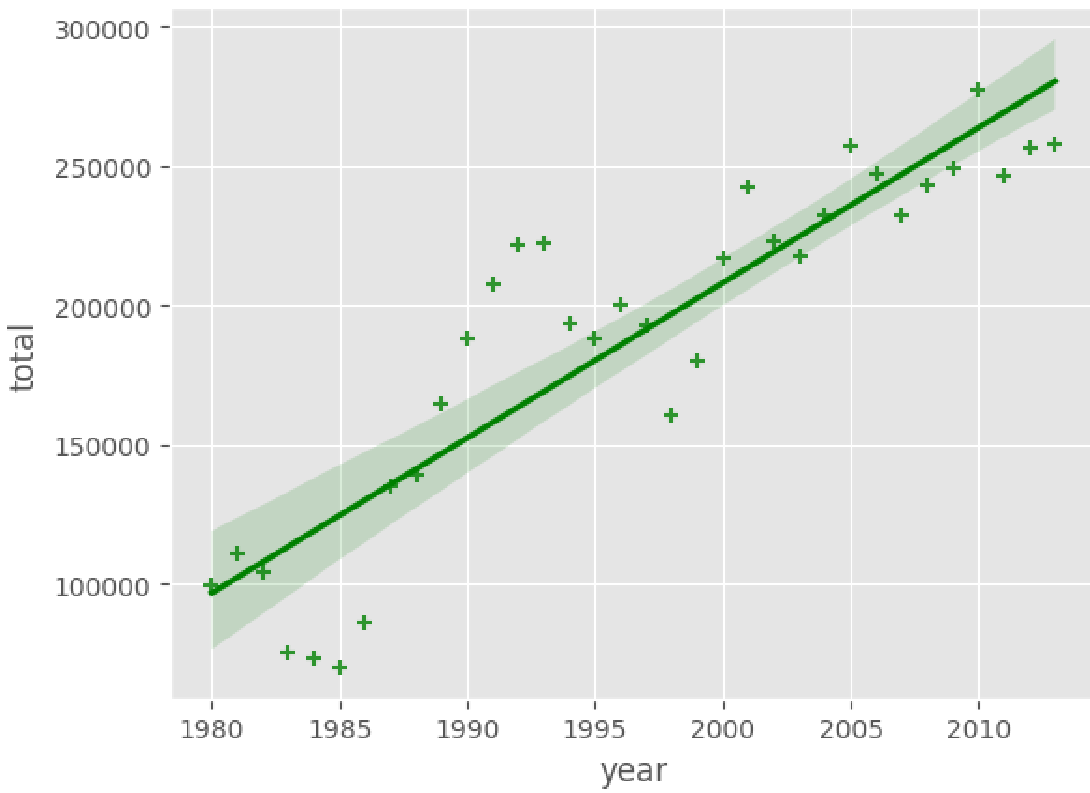
This is not magic; it is *seaborn*! You can also customize the color of the scatter plot and regression line. Let's change the color to green.

[45]:
```
sns.regplot(x='year', y='total', data=df_tot, color='green')
plt.show()
```



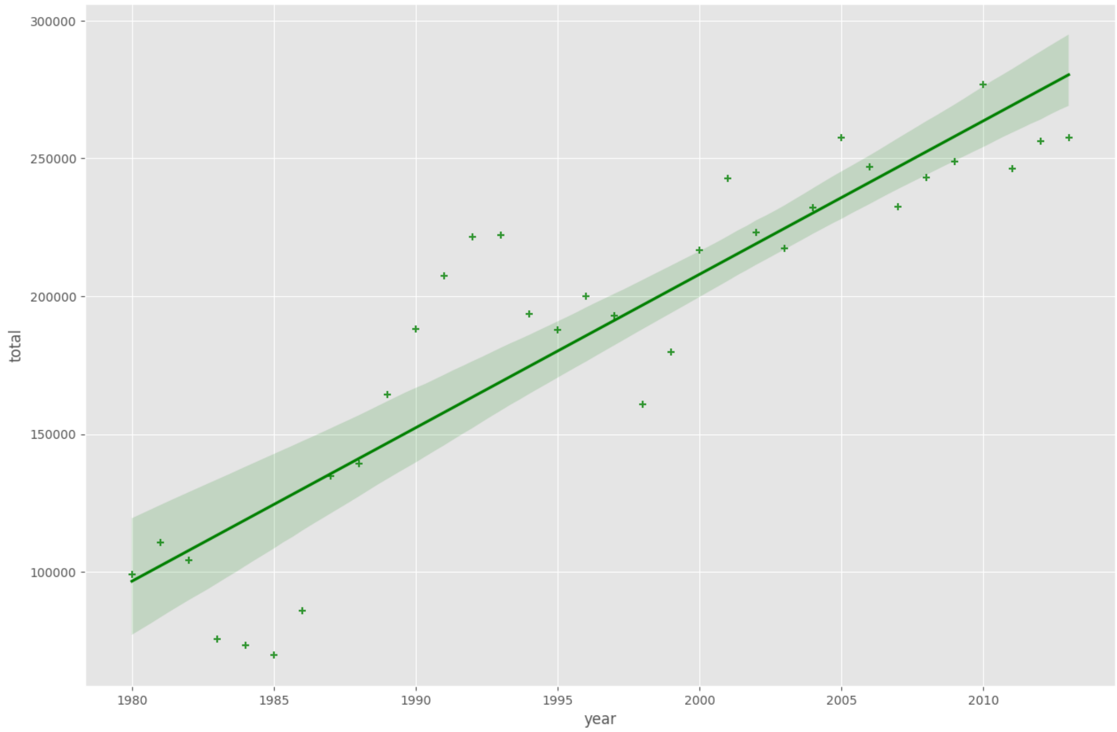
You can always customize the marker shape, so instead of circular markers, let's use `+`.

[46]:
```
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.

[47]:
```
plt.figure(figsize=(15, 10))
sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+')
plt.show()
```

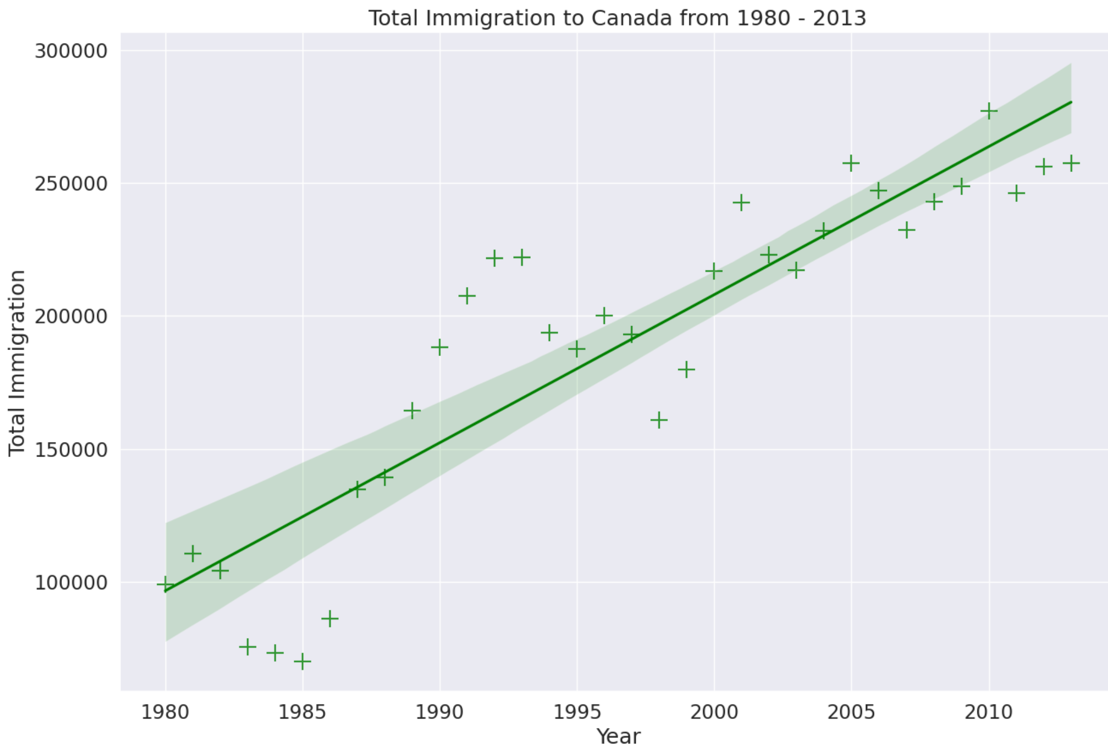


And let's increase the size of markers so they match the new size of the figure, and add a title and x- and y-labels.

[48]:
```
plt.figure(figsize=(15, 10))
ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+',␣
↪scatter_kws={'s': 200})

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!

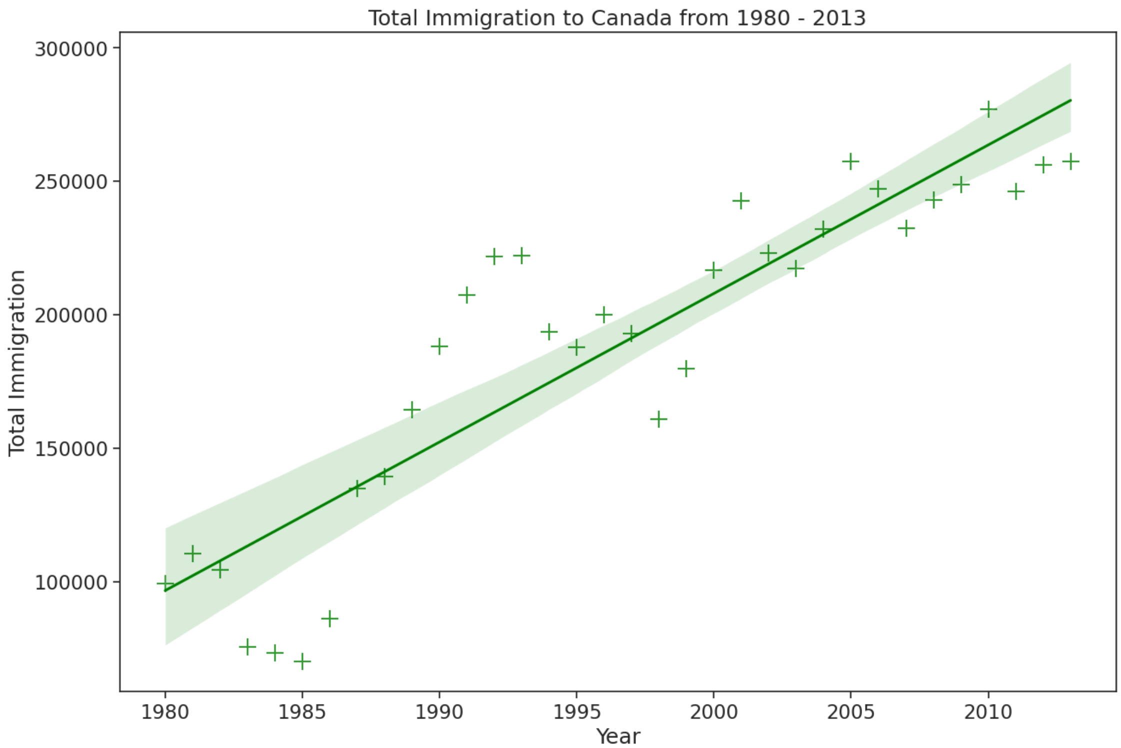
[49]:
```
plt.figure(figsize=(15, 10))

sns.set(font_scale=1.5)

ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='+',␣
 ↪scatter_kws={'s': 200})
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?

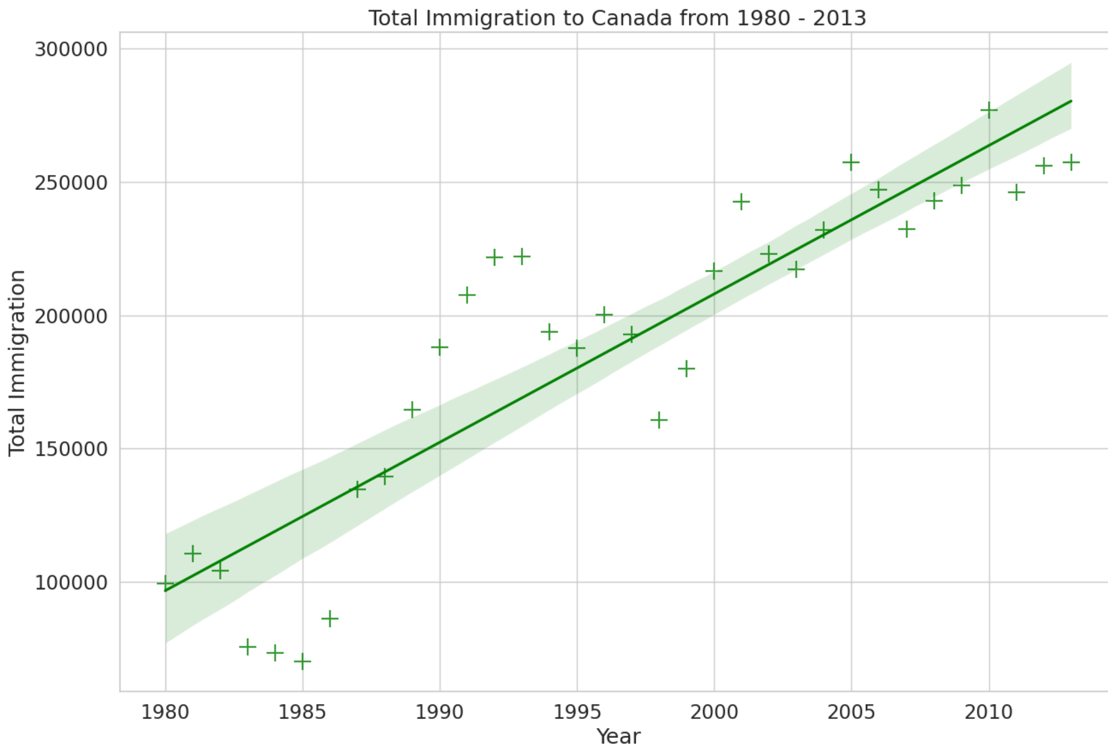
If you are not a big fan of the purple background, you can easily change the style to a white plain background.

[50]:
```
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='+',␣
 ↪scatter_kws={'s': 200})
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.

[51]:
```
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='+',␣
 ↪scatter_kws={'s': 200})
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.

[52]:
```
### type your answer here
```

Click here for a sample python solution

```
#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
ax.set(xlabel='Year', ylabel='Total Immigration')
ax.set_title('Total Immigrationn from Denmark, Sweden, and Norway to Canada from 1980 - 20
```

[ ]:

7.0.1 Thank you for completing this lab!

7.1 Author

Alex Aklson Dr. Pooja

##

<!–

7.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-07-07	2.7	Dr. Pooja	wordcloud, sns, piplite,pywaffle issue resolved
2023-06-11	2.6	Dr. Pooja	Clean data link, pywaffle,Categorical plots included
2021-05-19	2.3	Weiqing Wang	Fixed typos and code spells
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

##

[ ]:

[ ]:

[ ]: