**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**

1. [Exploring Datasets with *p*andas](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-4-1-Waffle-Charts-Word-Clouds-and-Regression-Plots-py-v2.0.ipynb#0)
2. [Downloading and Prepping Data](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-4-1-Waffle-Charts-Word-Clouds-and-Regression-Plots-py-v2.0.ipynb#2)
3. [Visualizing Data using Matplotlib](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-4-1-Waffle-Charts-Word-Clouds-and-Regression-Plots-py-v2.0.ipynb#4)
4. [Waffle Charts](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-4-1-Waffle-Charts-Word-Clouds-and-Regression-Plots-py-v2.0.ipynb#6)
5. [Word Clouds](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-4-1-Waffle-Charts-Word-Clouds-and-Regression-Plots-py-v2.0.ipynb#8)
6. [Regression Plots](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-4-1-Waffle-Charts-Word-Clouds-and-Regression-Plots-py-v2.0.ipynb#10)

**Exploring Datasets with *pandas* and Matplotlib**

Toolkits: The course heavily relies on [*pandas*](http://pandas.pydata.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) and [*Numpy*](http://www.numpy.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) for data wrangling, analysis, and visualization. The primary plotting library we will explore in the course is [Matplotlib](http://matplotlib.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

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?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) 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:

[1]:



**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:

[2]:



df\_can **=** pd.read\_excel(

'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.xlsx',

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

[ ]:



df\_can.head()

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

[ ]:



*# print the dimensions of the dataframe*

print(df\_can.shape)

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.

[3]:



*# clean up the dataset to remove unnecessary columns (eg. REG)*

df\_can.drop(['AREA','REG','DEV','Type','Coverage'], axis **=** 1, inplace **=** **True**)

​

*# let's rename the columns so that they make sense*

df\_can.rename (columns **=** {'OdName':'Country', 'AreaName':'Continent','RegName':'Region'}, inplace **=** **True**)

​

*# for sake of consistency, let's also make all column labels of type string*

df\_can.columns **=** list(map(str, df\_can.columns))

​

*# set the country name as index - useful for quickly looking up countries using .loc method*

df\_can.set\_index('Country', inplace **=** **True**)

​

*# add total column*

df\_can['Total'] **=** df\_can.sum (axis **=** 1)

​

*# years that we 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, 46)

**Visualizing Data using Matplotlib**

Import and setup matplotlib:

[4]:



**%**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: 3.4.2

**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.

[5]:



*# 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

[5]:

|  | **Continent** | **Region** | **DevName** | **1980** | **1981** | **1982** | **1983** | **1984** | **1985** | **1986** | **...** | **2013** | **Unnamed: 43** | **Unnamed: 44** | **Unnamed: 45** | **Unnamed: 46** | **Unnamed: 47** | **Unnamed: 48** | **Unnamed: 49** | **Unnamed: 50** | **Total** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Country** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Denmark** | Europe | Northern Europe | Developed regions | 272 | 293 | 299 | 106 | 93 | 73 | 93 | ... | 81 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 3901.0 |
| **Norway** | Europe | Northern Europe | Developed regions | 116 | 77 | 106 | 51 | 31 | 54 | 56 | ... | 59 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2327.0 |
| **Sweden** | Europe | Northern Europe | Developed regions | 281 | 308 | 222 | 176 | 128 | 158 | 187 | ... | 140 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 5866.0 |

3 rows × 46 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.

[15]:



*# 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})

[15]:

|  | **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.

[16]:



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

[20]:



*# 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})

[20]:

|  | **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.

[36]:



*# 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.

[37]:



waffle\_chart

[37]:

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.

[41]:



*# 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 432x288 with 0 Axes>

**Step 6.** Prettify the chart.

[42]:



*# 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.

[44]:



*# 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 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()

<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.

[48]:



**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:

[50]:



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.

[51]:



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](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.

[31]:



*# install wordcloud*

**!** pip3 install wordcloud

​

*# import package and its set of stopwords*

**from** wordcloud **import** WordCloud, STOPWORDS

​

print ('Wordcloud is installed and imported!')

Defaulting to user installation because normal site-packages is not writeable

Collecting wordcloud

Downloading wordcloud-1.8.1.tar.gz (220 kB)

|████████████████████████████████| 220 kB 4.6 MB/s eta 0:00:01

Requirement already satisfied: numpy>=1.6.1 in /Library/Python/3.8/site-packages (from wordcloud) (1.20.3)

Requirement already satisfied: pillow in /Users/weiqing/Library/Python/3.8/lib/python/site-packages (from wordcloud) (6.2.2)

Requirement already satisfied: matplotlib in /Library/Python/3.8/site-packages (from wordcloud) (3.4.2)

Requirement already satisfied: kiwisolver>=1.0.1 in /Library/Python/3.8/site-packages (from matplotlib->wordcloud) (1.3.1)

Requirement already satisfied: cycler>=0.10 in /Library/Python/3.8/site-packages (from matplotlib->wordcloud) (0.10.0)

Requirement already satisfied: python-dateutil>=2.7 in /Users/weiqing/Library/Python/3.8/lib/python/site-packages (from matplotlib->wordcloud) (2.8.1)

Requirement already satisfied: pyparsing>=2.2.1 in /Library/Python/3.8/site-packages (from matplotlib->wordcloud) (2.4.7)

Requirement already satisfied: six in /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.8/lib/python3.8/site-packages (from cycler>=0.10->matplotlib->wordcloud) (1.15.0)

Building wheels for collected packages: wordcloud

Building wheel for wordcloud (setup.py) ... done

Created wheel for wordcloud: filename=wordcloud-1.8.1-cp38-cp38-macosx\_10\_14\_6\_x86\_64.whl size=214077 sha256=bc45dc5891ea0ba52dea7dcd8bf0e6ec2279df10c31adbe6b28963c43eed129e

Stored in directory: /Users/weiqing/Library/Caches/pip/wheels/4d/3f/0d/a2ba9b7895c9f1be89018b3141c3df3d4f9c786c882ccfbc3b

Successfully built wordcloud

Installing collected packages: wordcloud

WARNING: The script wordcloud\_cli is installed in '/Users/weiqing/Library/Python/3.8/bin' which is not on PATH.

Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.

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.

[59]:



**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.

[56]:



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.

[60]:



*# 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)

[60]:

<wordcloud.wordcloud.WordCloud at 0x11bbc8880>

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

[61]:



*# 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.

[63]:



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.

[64]:



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*.

[67]:



*# 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.

[69]:



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.

[71]:



*# 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.

[72]:



df\_can.head()

[72]:

|  | **Continent** | **Region** | **DevName** | **1980** | **1981** | **1982** | **1983** | **1984** | **1985** | **1986** | **...** | **2013** | **Unnamed: 43** | **Unnamed: 44** | **Unnamed: 45** | **Unnamed: 46** | **Unnamed: 47** | **Unnamed: 48** | **Unnamed: 49** | **Unnamed: 50** | **Total** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Country** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Afghanistan** | Asia | Southern Asia | Developing regions | 16 | 39 | 39 | 47 | 71 | 340 | 496 | ... | 2004 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 58639.0 |
| **Albania** | Europe | Southern Europe | Developed regions | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ... | 603 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 15699.0 |
| **Algeria** | Africa | Northern Africa | Developing regions | 80 | 67 | 71 | 69 | 63 | 44 | 69 | ... | 4331 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 69439.0 |
| **American Samoa** | Oceania | Polynesia | Developing regions | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 6.0 |
| **Andorra** | Europe | Southern Europe | Developed regions | 0 | 0 | 0 | 0 | 0 | 0 | 2 | ... | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 15.0 |

5 rows × 46 columns

And what was the total immigration from 1980 to 2013?

[73]:



total\_immigration **=** df\_can['Total'].sum()

total\_immigration

[73]:

6409153.0

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

[79]:



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

[79]:

'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.

[80]:



*# create the word cloud*

wordcloud **=** WordCloud(background\_color**=**'white').generate(word\_string)

​

print('Word cloud created!')

Word cloud created!

[82]:



*# 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](https://seaborn.pydata.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) and more about *seaborn* regression plots by following this [link](http://seaborn.pydata.org/generated/seaborn.regplot.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01).

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*

[84]:



*# install seaborn*

**!** pip3 install seaborn

​

*# import library*

**import** seaborn **as** sns

​

print('Seaborn installed and imported!')

Defaulting to user installation because normal site-packages is not writeable

Requirement already satisfied: seaborn in /Library/Python/3.8/site-packages (0.11.1)

Requirement already satisfied: matplotlib>=2.2 in /Library/Python/3.8/site-packages (from seaborn) (3.4.2)

Requirement already satisfied: scipy>=1.0 in /Library/Python/3.8/site-packages (from seaborn) (1.6.3)

Requirement already satisfied: numpy>=1.15 in /Library/Python/3.8/site-packages (from seaborn) (1.20.3)

Requirement already satisfied: pandas>=0.23 in /Library/Python/3.8/site-packages (from seaborn) (1.2.4)

Requirement already satisfied: python-dateutil>=2.7 in /Users/weiqing/Library/Python/3.8/lib/python/site-packages (from matplotlib>=2.2->seaborn) (2.8.1)

Requirement already satisfied: pyparsing>=2.2.1 in /Library/Python/3.8/site-packages (from matplotlib>=2.2->seaborn) (2.4.7)

Requirement already satisfied: cycler>=0.10 in /Library/Python/3.8/site-packages (from matplotlib>=2.2->seaborn) (0.10.0)

Requirement already satisfied: kiwisolver>=1.0.1 in /Library/Python/3.8/site-packages (from matplotlib>=2.2->seaborn) (1.3.1)

Requirement already satisfied: pillow>=6.2.0 in /Users/weiqing/Library/Python/3.8/lib/python/site-packages (from matplotlib>=2.2->seaborn) (6.2.2)

Requirement already satisfied: six in /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.8/lib/python3.8/site-packages (from cycler>=0.10->matplotlib>=2.2->seaborn) (1.15.0)

Requirement already satisfied: pytz>=2017.3 in /Library/Python/3.8/site-packages (from pandas>=0.23->seaborn) (2021.1)

Seaborn installed and imported!

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

[85]:



*# 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()

[85]:

|  | **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.

[87]:



sns.regplot(x**=**'year', y**=**'total', data**=**df\_tot)

[87]:

<AxesSubplot: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.

[93]:



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 +.

[92]:



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.

[91]:



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.

[96]:



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!

[98]:



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.

[99]:



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.

[100]:



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.

[ ]:



*### 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**=**{'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!**[**¶**](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-4-1-Waffle-Charts-Word-Clouds-and-Regression-Plots-py-v2.0.ipynb#Thank-you-for-completing-this-lab!)