###############################################################

# Week 3.1: Waffle Charts, Word Clouds, and Regression Plots #

###############################################################

# importing libraries

import numpy as np

import pandas as pd

import xlrd

from PIL import Image # converts images into arrays

#############

# Read Data #

#############

May need this later for the regression analysis

# read url

df\_can = pd.read\_excel('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data\_Files/Canada.xlsx',

sheet\_name='Canada by Citizenship',

skiprows=range(20),

skipfooter=2)

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

# checking first 5 rows

df\_can.head()

df\_can.tail()

#######

# EDA #

#######

# checking dimensions

df\_can.shape

#############

# Reshaping #

#############

# dropping unnecessary columns

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

df\_can.head()

# renaming column headers

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

df\_can.head()

# new command for doing the same thing, ie checking column types

# tests if column is a string header, boolean logic

all(isinstance(column, str) for column in df\_can.columns)

# assigning column types to string, then rechecking

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

all(isinstance(column, str) for column in df\_can.columns)

# resetting index to unique country name

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

df\_can.head()

# preparing to visualize canadian data:

# by first adding a column for total immigration for 1980-2013

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

df\_can.head()

# printing m by n

# print('data dimensions:', df\_can.shape) # sick command for printing dimensionality

# creating a varlist that can be used to call colnmaes while plotting or subsetting

# capturing list of years, for category visualization

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

years

#####################################

# Visualizing Data using Matplotlib #

#####################################

# importing libs

import matplotlib as mpl

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches # required for waffle charts

mpl.style.use('ggplot')

# print('Matplotlib version: ', mpl.\_\_version\_\_) # >= 3.1.3

#################

# Waffle Charts #

#################

I can’t see any immediate use of waffle charts with this data. But if I do, I’ll be sure to incorporate it.

# creating a new dataframe comrpising of nordic countries

df\_dsn = df\_can.loc[['Denmark', 'Norway', 'Sweden'], :]

df\_dsn

#computing proportion of values to the whole

total\_values = sum(df\_dsn['Total'])

category\_proportions = [(float(value) / total\_values) for value in df\_dsn['Total']]

# printing these proportions for each value of total yearly immigration

# relative to total immigration from country to canada

"""

for i, proportion in enumerate(category\_proportions):

print (df\_dsn.index.values[i] + ': ' + str(proportion))

"""

# preparing to define the dimensions of our chart

width = 40 # width of chart

height = 10 # height of chart

total\_num\_tiles = width \* height # total number of tiles in grid

# print ('Total number of tiles is ', total\_num\_tiles)

# computing tiles per category ((country+year total)/country total)

tiles\_per\_category = [round(proportion \* total\_num\_tiles) for proportion in category\_proportions]

# printing out our number of tiles per category

"""

for i, tiles in enumerate(tiles\_per\_category):

print (df\_dsn.index.values[i] + ': ' + str(tiles))

"""

# declaring empty matrix to prepare for waffle chart

waffle\_chart = np.zeros((height, width))

# defining indices to loop through

category\_index = 0

tile\_index = 0

# populating the waffle chart

for col in range(width):

for row in range(height):

tile\_index += 1

# if the number for current category is equal to corresponding tiles..

if tile\_index > sum(tiles\_per\_category[0:category\_index]):

# .. proceed to next

category\_index += 1

# setting class value to an integer

waffle\_chart[row, col] = category\_index

# print ('Waffle chart populated!')

waffle\_chart

# preparing to map waffle to a visual

"""

fig = plt.figure()

# using matshow to display the waffle chart

colormap = plt.cm.coolwarm

plt.matshow(waffle\_chart, cmap=colormap)

plt.colorbar()

plt.show()

"""

# prettyfying or souping up the chart, iteration 1

# declaring a new figure object

"""

fig = plt.figure()

# using matshow to display the waffle chart

colormap = plt.cm.coolwarm

plt.matshow(waffle\_chart, cmap=colormap)

plt.colorbar()

# getting the axes

ax = plt.gca()

# setting minor ticks

ax.set\_xticks(np.arange(-.5, (width), 1), minor=True)

ax.set\_yticks(np.arange(-.5, (height), 1), minor=True)

# adding in gridlines based on minor ticks

ax.grid(which='minor', color='w', linestyle='-', linewidth=2)

plt.xticks([])

plt.yticks([])

plt.show()

"""

# prettyfying, souping up the chart, iteration 2

# declaring a new figure object

"""

fig = plt.figure()

# using matshow to display the waffle chart

colormap = plt.cm.coolwarm

plt.matshow(waffle\_chart, cmap=colormap)

plt.colorbar()

# getting the axes

ax = plt.gca()

# setting minor ticks

ax.set\_xticks(np.arange(-.5, (width), 1), minor=True)

ax.set\_yticks(np.arange(-.5, (height), 1), minor=True)

# adding in gridlines based on minor ticks

ax.grid(which='minor', color='w', linestyle='-', linewidth=2)

plt.xticks([])

plt.yticks([])

# computing cumulative sums

values\_cumsum = np.cumsum(df\_dsn['Total'])

total\_values = values\_cumsum[len(values\_cumsum) - 1]

# creating the 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))

# adding 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()

# awesome, the legend really adds a cool element

"""

# creating instead a function that generates our waffle chart

def create\_waffle\_chart(categories, values, height, width, colormap, value\_sign=''):

#computing proportion of values to the whole

total\_values = sum(values)

category\_proportions = [(float(value) / total\_values) for value in values]

# total number of tiles in grid

total\_num\_tiles = width \* height

print ('Total number of tiles is ', total\_num\_tiles)

# computing tiles per category ((country+year total)/country total)

tiles\_per\_category = [round(proportion \* total\_num\_tiles) for proportion in category\_proportions]

# printing out our number of tiles per category

for i, tiles in enumerate(tiles\_per\_category):

print (df\_dsn.index.values[i] + ': ' + str(tiles))

# declaring empty matrix to prepare for waffle chart

waffle\_chart = np.zeros((height, width))

# defining indices to loop through

category\_index = 0

tile\_index = 0

# populating the waffle chart

for col in range(width):

for row in range(height):

tile\_index += 1

# if the number for current category is equal to corresponding tiles..

if tile\_index > sum(tiles\_per\_category[0:category\_index]):

# .. proceed to next

category\_index += 1

# setting class value to an integer

waffle\_chart[row, col] = category\_index

# declaring a new figure object

fig = plt.figure()

# using matshow to display the waffle chart

colormap = plt.cm.coolwarm

plt.matshow(waffle\_chart, cmap=colormap)

plt.colorbar()

# getting the axes

ax = plt.gca()

# setting minor ticks

ax.set\_xticks(np.arange(-.5, (width), 1), minor=True)

ax.set\_yticks(np.arange(-.5, (height), 1), minor=True)

# adding in gridlines based on minor ticks

ax.grid(which='minor', color='w', linestyle='-', linewidth=2)

plt.xticks([])

plt.yticks([])

# computing cumulative sums

values\_cumsum = np.cumsum(values)

total\_values = values\_cumsum[len(values\_cumsum) - 1]

# creating the 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))

# adding 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

width = 40 # width of our chart

height = 10 # height of our chart

categories = df\_dsn.index.values # categories: of year\_country

values = df\_dsn['Total'] # corresponding values to year\_country categories

colormap = plt.cm.coolwarm # color map class being added

# create\_waffle\_chart(categories, values, height, width, colormap)

###############

# Word Clouds #

###############

# importing package

from wordcloud import WordCloud, STOPWORDS

import wget

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

# downloading text file using wget

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data\_Files/alice\_novel.txt'

# wget.download(url, 'alice\_novel.txt')

# opening text file and reading it into a variable called alice\_novel

alice\_novel = open('alice\_novel.txt', 'r').read()

# print('File downloaded and saved!')

# removes redundant 'stopwords'

stopwords = set(STOPWORDS)

# preparing to generate a word cloud

# by first subsetting to the first 2000 words of the novel

# declaring a word cloud object

alice\_wc = WordCloud(

background\_color='white',

max\_words=2000,

stopwords=stopwords

)

# generating the word cloud

alice\_wc.generate(alice\_novel)

"""

# actually plotting the cloud display, iteration 1

plt.imshow(alice\_wc, interpolation='bilinear')

plt.axis('off')

plt.show()

"""

"""

# resizing the word cloud, iteration 2

fig = plt.figure()

fig.set\_figwidth(14) # setting width

fig.set\_figheight(18) # setting height

# displaying the cloud, iteration 2

plt.imshow(alice\_wc, interpolation='bilinear')

plt.axis('off')

plt.show()

"""

"""

# scrapping unnecessary words, iteration 3

stopwords.add('said') # adding said to stopwords

# re-generating the cloud visual, iteration 3

alice\_wc.generate(alice\_novel)

# displaying the word cloud, iteration 3

fig = plt.figure()

fig.set\_figwidth(14) # setting width

fig.set\_figheight(18) # setting height

plt.imshow(alice\_wc, interpolation='bilinear')

plt.axis('off')

plt.show()

"""

# using wordcloud to superimpose cloud visuals onto a mask image

# downloading image

# url2 = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Images/alice\_mask.png'

# wget.download(url2, 'alice\_mask.png')

# reading in and saving mask image file

alice\_mask = np.array(Image.open('alice\_mask.png'))

# print('Image downloaded and saved!')

"""

# opening and displaying image shell

fig = plt.figure()

fig.set\_figwidth(14) # setting width

fig.set\_figheight(18) # setting height

plt.imshow(alice\_mask, cmap=plt.cm.gray, interpolation='bilinear')

plt.axis('off')

plt.show()

"""

"""

# declaring a word cloud object, iteration 4

alice\_wc = WordCloud(background\_color='white', max\_words=2000, mask=alice\_mask, stopwords=stopwords)

# generating the word cloud, iteration 4

alice\_wc.generate(alice\_novel)

# displaying the word cloud, iteration 4

fig = plt.figure()

fig.set\_figwidth(14) # setting width

fig.set\_figheight(18) # setting height

plt.imshow(alice\_wc, interpolation='bilinear')

plt.axis('off')

plt.show()

"""

# switching gears back to regression plots, and canada data

df\_can.head()

# checking total immigration from 190 to 2013

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

total\_immigration

# using single word country names to generate word cloud

# based on frequency

max\_words = 90

word\_string = ''

for country in df\_can.index.values:

# checking if a country's name is a single-word name

if len(country.split(' ')) == 1:

repeat\_num\_times = int(df\_can.loc[country, 'Total']/float(total\_immigration)\*max\_words)

word\_string = word\_string + ((country + ' ') \* repeat\_num\_times)

# displaying generated text

word\_string

"""

# creating the word cloud, no stopwords

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

# print('Word cloud created!')

# displaying the country word cloud

fig = plt.figure()

fig.set\_figwidth(14) # setting width

fig.set\_figheight(18) # setting height

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.show()

"""

####################

# Regression Plots #

####################

# importing seaborn

import seaborn as sns

# print('Seaborn installed and imported!')

# using the sum() method to get the total population per year

# in essence, collapsing by year,across all countries

df\_tot = pd.DataFrame(df\_can[years].sum(axis=0))

# changing year type to float to measure incremental change

df\_tot.index = map(float, df\_tot.index)

# resetting index to use as a column

df\_tot.reset\_index(inplace=True)

# renaming columns

df\_tot.columns = ['year', 'total']

# viewing outputted dataframe

df\_tot.head()

"""

# plot 1, using regplot

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

plt.show()

"""

"""

# plot 2, switching color to green

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

plt.show()

"""

"""

# plot 3, changing markers to + signs

ax = sns.regplot(x='year', y='total', data=df\_tot, color='green', marker='+')

plt.show()

"""

"""

# plot 4, blowing up the plot size

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

ax = sns.regplot(x='year', y='total', data=df\_tot, color='green', marker='+')

plt.show()

"""

"""

# plot 5, blowing up marker size, adding axis labels, title

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') # adding axes labels

ax.set\_title('Total Immigration to Canada from 1980 - 2013') # add title

plt.show()

"""

"""

# plot 6, increasing font size of tickmark labels, title, axes

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') # adding axes labels

ax.set\_title('Total Immigration to Canada from 1980 - 2013') # add title

plt.show()

"""

"""

# plot 6.1, optional features: white background

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

sns.set(font\_scale=1.5)

sns.set\_style('ticks') # changing background to white

ax = sns.regplot(x='year', y='total', data=df\_tot, color='green', marker='+', scatter\_kws={'s': 200})

ax.set(xlabel='Year', ylabel='Total Immigration') # adding axes labels

ax.set\_title('Total Immigration to Canada from 1980 - 2013') # add title

plt.show()

"""

"""

# plot 6.2, optional features: gridlines

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') # adding axes labels

ax.set\_title('Total Immigration to Canada from 1980 - 2013') # add title

plt.show()

"""

############################################################

# Q1: Scatterplot of Nordic Countries, fitted with Regline #

############################################################

# 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.

# subsetting to denmark, norway, and sweden

df\_dsn = df\_can.loc[['Denmark', 'Norway', 'Sweden'], :]

# collapsing across countries, for each year

dsn\_tot = pd.DataFrame(df\_dsn[years].sum(axis=0))

# changing year type to float to measure incremental change

dsn\_tot.index = map(float, dsn\_tot.index)

# resetting index to use as a column

dsn\_tot.reset\_index(inplace=True)

# renaming columns

dsn\_tot.columns = ['year', 'total']

# viewing outputted dataframe

dsn\_tot.head()

"""

# creating a fleshed out regression plot for our data

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

sns.set(font\_scale=1.5)

ax = sns.regplot(x='year', y='total', data=dsn\_tot, color='green', marker='+', scatter\_kws={'s': 200})

ax.set(xlabel='Year', ylabel='Total Immigration') # adding axes labels

ax.set\_title('Total Immigration from Nordic Countries to Canada from 1980 - 2013') # add title

plt.show()

"""

# in order to display plot within window

# plt.show()