basic_plot_4

February 3, 2023

[ ]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

[ ]:
```
# import module to convert images into arrays
from PIL import Image
```

[ ]:
```
# for waffle charts
import matplotlib.patches as mpat
```

[ ]:
```
# read the data file
df_can = pd.read_excel ('Canada.xlsx', sheet_name = 'Canada by Citizenship',␣
 ↪skiprows = 20, skipfooter = 2)
```

[ ]:
```
# get the dimension of the data frame
df_can.shape
```

[ ]:
```
# get the head of the data frame
df_can.head()
```

[ ]:
```
# 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)

# years that we will be using in this lesson - useful for plotting later on
```

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

# add the number of immigrants for all the years for each country
df_tot = df_can[years].sum(axis = 1)

# create a new column
df_can ['Total'] = df_tot
```

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.

[ ]:
```
# create a data frame for Denmark, Norway, and Sweden
df_nord = df_can.loc[['Denmark', 'Norway', 'Sweden'], :]

# check the head
df_nord.head()
```

[ ]:
```
# compute the proportion of each country with respect to the total
tot_val = df_nord['Total'].sum(axis = 0)

print (tot_val)

country_proportion = df_nord['Total'] / tot_val

print (country_proportion)
```

[ ]:
```
# specify the overall size of the waffle chart
width = 40
height = 10

# total number of tiles
num_tiles = width * height

print (num_tiles)
```

[ ]:
```
# compute the number of tiles for each country
tiles_country = (country_proportion * num_tiles).round().astype(int)

# print the number of tiles per country
print (tiles_country)
```

[ ]:
```
# create a 2-D matrix to simulate the waffle
waffle_chart = np.zeros ((height, width), dtype = np.uint)

# define indices to loop through the waffle chart
country_index = 0
```

```
tile_index = 0

# populate the waffle chart
for col in range (width):
  for row in range (height):
    tile_index += 1

    # check if the number of tiles populated is equal to the
    # allocated tiles for that country
    if (tile_index > sum(tiles_country[0:country_index])):
      country_index += 1  # go to the next county

    # fill the waffle_chart
    waffle_chart[row,col] = country_index

# print waffle_chart
print (waffle_chart)
```

[ ]:
```
# make the waffle_chart into a visual
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 the minor ticks
ax.set_xticks (np.arange (-.5, (width), 1), minor = True)
ax.set_yticks (np.arange (-.5, (height), 1), minor = True)

# add grid lines based on minor ticks
ax.grid (which = 'minor', color = 'w', linestyle = '-', linewidth = 2)

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

# compute cummulative sum of individual countries to match
# color schemes between chart and legend
val_sum = np.cumsum (df_nord['Total'])
tot_val = val_sum [len(val_sum) - 1]

# create a legend
legend_handles = []
for i, country in enumerate (df_nord.index.values):
```

```
  label_str = country + '(' + str(df_nord['Total'][i]) + ')'
  color_val = colormap (float(val_sum[i]) / tot_val)
  legend_handles.append (mpat.Patch(color = color_val, label = label_str))

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

plt.show()
```

Create Waffle Chart Generation Function

The parameters of that function are: * categories: unique categories or classes in the dataframe * values: values corresponding to categories or classes * height: defined height of waffle chart * width: defined width of waffle chart * colormap: colormap class * value_sign: has default value of empty string

[ ]:
```
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 (categories[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(mpat.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()
```

[ ]:
```
# create the parameters for the waffle chart generator
width = 40 # width of chart
height = 10 # height of chart

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

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

[ ]:
```
# now run the function
create_waffle_chart(categories, values, height, width, colormap)
```

Regression Line with Seaborn

[ ]:
```
# import seaborn
import seaborn as sb
```

[ ]:
```
# years that we will be using in this lesson - useful for plotting later on
years = list(map(str, range(1980, 2014)))

# add the total population per year for each country
df_tot = pd.DataFrame(df_can[years].sum(axis = 0))

# change the years to type float
df_tot.index = map (float, df_tot.index)

# reset the index
df_tot.reset_index (inplace = True)

# rename columns
df_tot.columns = ['year', 'total']

# view head
df_tot.head()
```

[ ]:
```
# draw the regression line
plt.figure (figsize = (15,10))
```

```
sb.set (font_scale = 1.5)

sb.set_style ('whitegrid')

ax = sb.regplot (x = 'year', y = 'total', data = df_tot, color = 'b', marker =␣
↪'o')
ax.set (xlabel = 'Year', ylabel = 'Total Immigration')
ax.set_title ('Total Immigration to Canada 1980 - 2013')
plt.show()
```

[ ]:

[ ]: