**Week 3 - Advanced Visualizations and Geospatial Data**

* Waffle Charts (Not included in Matplotlib) (OR use **PyWaffle**)
  + - Display progress towards goals
    - TileChart
    - Example:
      * Sample Prep
        + # 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**
      * **Step 1.** The first step into creating a waffle chart is determing the proportion of each category with respect to the total.
        + # 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})**
      * **Step 2.** The second step is defining the overall size of the waffle chart.
        + **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}.')**
      * **Step 3.** The third step is using the proportion of each category to determine it respective number of tiles
        + **# 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})**
      * **Step 4.** The fourth step is creating a matrix that resembles the waffle chart and populating it.
        + **# 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!')**
      * **Step 5.** Map the waffle chart matrix into a visual.
        + **# 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()**
      * **Step 6.** Prettify the chart.
        + **# 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()**
* Word Clouds (Not included in Matplotlib)
  + - Display different word frequency
    - Example:
      * <https://github.com/amueller/word_cloud/>
      * Use this to install package
        + **# install wordcloud**
        + **!pip3 install wordcloud==1.8.1**
        + **# import package and its set of stopwords**
        + **from wordcloud import WordCloud, STOPWORDS**
        + **print ('Wordcloud is installed and imported!')**
    - Import Text
      * **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")**
    - Remove redundant stop words
      * **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.
      * **# 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)**
    - Awesome! Now that the word cloud is created, let's visualize it.
      * **# display the word cloud**
      * **fig = plt.figure(figsize=(14, 18))**
      * **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.
      * **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.
      * **# 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.
      * **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.
      * **# 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()**
* Seaborn and Regression Plots
  + - Seaborn
      * **Import seaborn as sns**
      * **Ax = sns.regplot (x= ‘year’, y=’total’, color = ‘green’, marker = ‘+’, data = df\_tot)**
      * Visualization based on Matplotlib
      * Code is much more efficient in comparison to matplotlib.
    - Example:
      * **Import seaborn as sns**
    - Get Data frame
      * # 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']**
      * **#** set data to type int as necessary for plotting
      * **df\_tot['year'] = df\_total['year'].astype(int)**
      * # view the final dataframe
      * **df\_tot.head()**
    - With seaborn, generating a regression plot is as simple as calling the **regplot** function.
      * **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 with Answer**

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.

**df\_dns = df\_can.loc[['Denmark', 'Sweden', 'Norway'], years].transpose()**

**df\_total = pd.DataFrame(df\_dns.sum(axis=1))**

**df\_total.reset\_index(inplace = True)**

**df\_total.columns = ['Year', 'Total']**

**df\_total['Year'] = df\_total['Year'].astype(int)**

**#df\_total.head()**

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

**sns.set(font\_scale = 1.5)**

**sns.set\_style('whitegrid')**

**ax = sns.regplot(x = 'Year', y = 'Total', data = df\_total, color = 'red', scatter\_kws={'s':200})**

**Chart, scatter chart

Description automatically generated**

* + Introduction to Folium and Map Styles
    - Folium is a python library that helps you create several types of leaflet maps.
    - Example (imports)
      * **import numpy as np** # useful for many scientific computing in Python
      * **import pandas as pd** # primary data structure library
      * **import folium**
    - Generating the world map is straightforward in **Folium**. You simply create a **Folium** Map object, and then you display it. What is attractive about **Folium** maps is that they are interactive, so you can zoom into any region of interest despite the initial zoom level.
      * # define the world map
      * **world\_map = folium.Map(location=[56.130, -106.35], zoom\_start=4)**
      * # display world map
      * **world\_map**
  + Maps with Markers
    - A. Stamen Toner Maps
      * # create a Stamen Toner map of the world centered around Canada
      * **world\_map = folium.Map(location=[56.130, -106.35], zoom\_start=4, tiles='Stamen Toner')**
      * # display map
      * **world\_map**
    - B. Stamen Terrain Maps
      * # create a Stamen Toner map of the world centered around Canada
      * **world\_map = folium.Map(location=[56.130, -106.35], zoom\_start=4, tiles='Stamen Terrain')**
      * # display map
      * **world\_map**
  + Choropleth Maps
    - Downloading Data
      * **df\_incidents = pd.read\_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Police\_Department\_Incidents\_-\_Previous\_Year\_\_2016\_.csv')**
      * **print('Dataset downloaded and read into a pandas dataframe!')**
    - Let's find out how many entries there are in our dataset.
      * **df\_incidents.shape**
    - So the dataframe consists of 150,500 crimes, which took place in the year 2016. In order to reduce computational cost, let's just work with the first 100 incidents in this dataset.
      * # get the first 100 crimes in the df\_incidents dataframe
      * **limit = 100**
      * **df\_incidents = df\_incidents.iloc[0:limit, :]**
      * **df\_incidents.head()**
    - Now that we reduced the data a little, let's visualize where these crimes took place in the city of San Francisco. We will use the default style, and we will initialize the zoom level to 12.
      * # San Francisco latitude and longitude values
      * **latitude = 37.77**
      * **longitude = -122.42**
      * # create map and display it
      * **sanfran\_map = folium.Map(location=[latitude, longitude], zoom\_start=12)**
      * # display the map of San Francisco
      * **sanfran\_map**
    - Now let's superimpose the locations of the crimes onto the map. The way to do that in **Folium** is to create a feature group with its own features and style and then add it to the sanfran\_map.
      * # instantiate a feature group for the incidents in the dataframe
      * **incidents = folium.map.FeatureGroup()**
      * # loop through the 100 crimes and add each to the incidents feature group
      * **for lat, lng, in zip(df\_incidents.Y, df\_incidents.X):**
      * **incidents.add\_child(**
      * **folium.features.CircleMarker(**
      * **[lat, lng],**
      * **radius=5, # define how big you want the circle markers to be**
      * **color='yellow',**
      * **fill=True,**
      * **fill\_color='blue',**
      * **fill\_opacity=0.6**
      * **)**
      * **)**
      * **# add pop-up text to each marker on the map**
      * **latitudes = list(df\_incidents.Y)**
      * **longitudes = list(df\_incidents.X)**
      * **labels = list(df\_incidents.Category)**
      * **for lat, lng, label in zip(latitudes, longitudes, labels):**
      * **folium.Marker([lat, lng], popup=label).add\_to(sanfran\_map)**
      * # add incidents to map
      * **sanfran\_map.add\_child(incidents)**
    - If you find the map to be so congested will all these markers, there are two remedies to this problem. The simpler solution is to remove these location markers and just add the text to the circle markers themselves as follows:
      * # create map and display it
      * **sanfran\_map = folium.Map(location=[latitude, longitude], zoom\_start=12)**
      * # loop through the 100 crimes and add each to the map
      * **for lat, lng, label in zip(df\_incidents.Y, df\_incidents.X, df\_incidents.Category):**
      * **folium.features.CircleMarker(**
      * **[lat, lng],**
      * **radius=5, # define how big you want the circle markers to be**
      * **color='yellow',**
      * **fill=True,**
      * **popup=label,**
      * **fill\_color='blue',**
      * **fill\_opacity=0.6**
      * **).add\_to(sanfran\_map)**
      * # show map
      * **sanfran\_map**
    - To implement this, we start off by instantiating a MarkerCluster object and adding all the data points in the dataframe to this object.
      * **from folium import plugins**
      * # let's start again with a clean copy of the map of San Francisco
      * **sanfran\_map = folium.Map(location = [latitude, longitude], zoom\_start = 12)**
      * # instantiate a mark cluster object for the incidents in the dataframe
      * **incidents = plugins.MarkerCluster().add\_to(sanfran\_map)**
      * # loop through the dataframe and add each data point to the mark cluster
      * **for lat, lng, label, in zip(df\_incidents.Y, df\_incidents.X, df\_incidents.Category):**
      * **folium.Marker(**
      * **location=[lat, lng],**
      * **icon=None,**
      * **popup=label,**
      * **).add\_to(incidents)**
      * # display map
      * **sanfran\_map**

**Density Map (**Choropleth)

* In order to create a Choropleth map, we need a GeoJSON file that defines the areas/boundaries of the state, county, or country that we are interested in. In our case, since we are endeavoring to create a world map, we want a GeoJSON that defines the boundaries of all world countries. For your convenience, we will be providing you with this file, so let's go ahead and download it. Let's name it **world\_countries.json**.
  + # download countries geojson file
  + **! wget --quiet https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/world\_countries.json**
  + **print('GeoJSON file downloaded!')**
* Now that we have the GeoJSON file, let's create a world map, centered around **[0, 0]** latitude and longitude values, with an initisal zoom level of 2.
  + **world\_geo = r'world\_countries.json' # geojson file**
  + **# create a plain world map**
  + **world\_map = folium.Map(location=[0, 0], zoom\_start=2)**
* And now to create a Choropleth map, we will use the *choropleth* method with the following main parameters:
  1. geo\_data, which is the GeoJSON file.
  2. data, which is the dataframe containing the data.
  3. columns, which represents the columns in the dataframe that will be used to create the Choropleth map.
  4. key\_on, which is the key or variable in the GeoJSON file that contains the name of the variable of interest. To determine that, you will need to open the GeoJSON file using any text editor and note the name of the key or variable that contains the name of the countries, since the countries are our variable of interest. In this case, **name** is the key in the GeoJSON file that contains the name of the countries. Note that this key is case\_sensitive, so you need to pass exactly as it exists in the GeoJSON file.
* # generate choropleth map using the total immigration of each country to Canada from 1980 to 2013

**world\_geo = r'world\_countries.json'**

# create a numpy array of length 6 and has linear spacing from the minimum total immigration to the maximum total immigration

**threshold\_scale = np.linspace(df\_can['Total'].min(),**

**df\_can['Total'].max(),**

**6, dtype=int)**

**threshold\_scale = threshold\_scale.tolist()** # change the numpy array to a list

**threshold\_scale[-1] = threshold\_scale[-1] + 1** # make sure that the last value of the list is greater than the maximum immigration

# let Folium determine the scale.

**world\_map = folium.Map(location=[0, 0], zoom\_start=2)**

**world\_map.choropleth(**

**geo\_data=world\_geo,**

**data=df\_can,**

**columns=['Country', 'Total'],**

**key\_on='feature.properties.name',**

**threshold\_scale=threshold\_scale,**

**fill\_color='YlOrRd',**

**fill\_opacity=0.7,**

**line\_opacity=0.2,**

**legend\_name='Immigration to Canada',**

**reset=True**

**)**

**world\_map**

* + Lab: Advanced Visualization Tools
  + Lab: Creating Maps and Visualizing Geospatial Data
  + Quiz: Advanced Visualization Tools
  + Quiz: Visualizing Geospatial Data
  + Peer-review Assignment