Lab Manual for Tools and Technologies for Data Science

Lab-06

Data Exploration with advanced matplotlib

Table of Contents

Lab 6: Data Exploration with advanced matplotlib

1. Objective

- Word Clouds
- Mask Images
- Folium Maps
- World Maps

2. 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. WordCloud is a technique to show which words are the most frequent among the given text.

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.

```
pip install wordcloud
```

Then, before using any functions is check out the docstring of the function, and see all required and optional arguments. To do so, type ?function and run it to get all information. ?wordcloud

# Create and generate a word cloud image:
wordcloud = WordCloud().generate(provide text)

```
from wordcloud import WordCloud, STOPWORDS



hunger_game = open('hunger-game.txt', 'r').read()
stopwords = set(STOPWORDS)
# instantiate a word cloud object
hunger_wc = WordCloud(
    background_color='white',
    max_words=500,
    stopwords=stopwords
)

# generate the word cloud
hunger_wc.generate(hunger_game)
```

#display the word cloud

```
plt.imshow(hunger_wc)
plt.axis('off')
plt.show()
```

However, Us isn't really an informative word. So let's add it to our stopwords and re-generate the cloud.

```
stopwords.add('us') # add the words said to stopwords

# re-generate the word cloud
hunger_wc.generate(hunger_game)

# display the cloud
fig = plt.figure()
fig.set_figwidth(14) # set width
fig.set_figheight(18) # set height

plt.imshow(hunger_wc)
plt.axis('off')
plt.show()
```

Now, let's map these words into alice of wonderland!
Seriously,
Even a bottle of wine if you wish!

```
from PIL import Image
game_mask = np.array(Image.open('alice.png'))


fig = plt.figure()
fig.set_figwidth(14) # set width
fig.set_figheight(18) # set height

plt.imshow(game_mask, cmap=plt.cm.gray, interpolation='bilinear')
plt.axis('off')
plt.show()
```

```
game_wc = WordCloud(background_color='white', max_words=100, mask=game_mas
k, stopwords=stopwords)

# generate the word cloud
```

```
game_wc.generate(hunger_game)

# display the word cloud
fig = plt.figure()
fig.set_figwidth(8) # set width
fig.set_figheight(10) # set height

plt.imshow(game_wc, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Seaborn: Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

Let's Create a new data frame that stores that total number of landed immigrants to Canada per year from 1980 to 2013.

```
# we can use the sum() method to get the total population per year
df_tot = pd.DataFrame(df[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()

ax = sns.regplot(x='year', y='total', data=df_tot, color='green', marker='
+')
```

```
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}) #scatter_kws adjust the size of markers

ax.set(xlabel='Year', ylabel='Total Immigration') # add x- and y-labels
ax.set_title('Total Immigration to Canada from 1980 - 2013') # add title
```

3. Folium

Folium is a powerful Python library that helps you create several types of Leaflet maps. The fact that the Folium results are interactive makes this library very useful for dashboard building.

Import folium

```
# define the world map
world_map = folium.Map()

# display world map
world_map
```

Go ahead. Try zooming in and out of the rendered map above.

You can customize this default definition of the world map by specifying the centre of your map and the initial zoom level. All locations on a map are defined by their respective Latitude and Longitude values. So, you can create a map and pass in a center of Latitude and Longitude values of [0, 0]. For a defined center, you can also define the initial zoom level into that location when the map is rendered. The higher the zoom level the more the map is zoomed into the center.

Let's create a map centered around Canada and play with the zoom level to see how it affects the rendered map.

```
# define the world map centered around Canada with a low zoom level
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4)

# display world map
world_map
```

Question: Create a map of Pakistan with a zoom level of 4.

```
world_map = folium.Map(location=[30, 70], zoom_start=8)

# display world map
world_map
```

4. Map Types

Another cool feature of Folium is that you can generate different map styles.

A. Stamen Toner Maps

These are high-contrast B+W (black and white) maps. They are perfect for data mashups and exploring river meanders and coastal zones.

Let's create a Stamen Toner map of Pakistan with a zoom level of 4.

```
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4, tiles='Stamen Toner')
```

B. Stamen Terrain Maps

These are maps that feature hill shading and natural vegetation colors. They showcase advanced labeling and linework generalization of dual-carriageway roads. Let's create a Stamen Terrain map with zoom level 4.

```
world_map = folium.Map(location=[30, 70], zoom_start=8, tiles='Stamen Terrain')
```