Untitled6

August 4, 2019

[1]:

```
import numpy as np # library to handle data in a vectorized manner

import pandas as pd # library for data analsysis
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

import json # library to handle JSON files

!pip install geopy
from geopy.geocoders import Nominatim # convert an address into latitude and longitude values

import requests # library to handle requests
from pandas.io.json import json_normalize # tranform JSON file into a pandas dataframe

# Matplotlib and associated plotting modules
import matplotlib.cm as cm
import matplotlib.colors as colors

# import k-means from clustering stage
from sklearn.cluster import KMeans

!conda install -c conda-forge folium=0.5.0 --yes
import folium # map rendering library

print('Libraries imported.')
```

```
Collecting geopy
  Downloading https://files.pythonhosted.org/packages/80/93/d384479da0ead7
12bdaf697a8399c13a9a89bd856ada5a27d462fb45e47b/geopy-1.20.0-py2.py3-none-any.whl
(100kB)
     || 102kB 17.1MB/s ta 0:00:01
Collecting geographiclib<2,>=1.49 (from geopy)
  Downloading https://files.pythonhosted.org/packages/5b/ac/4f348828091490d77899
bc74e92238e2b55c59392f21948f296e94e50e2b/geographiclib-1.49.tar.gz
Building wheels for collected packages: geographiclib
  Building wheel for geographiclib (setup.py) ... done
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/99/45/d1/1495479
```

```
7e2a976083182c2e7da9b4e924509e59b6e5c661061
Successfully built geographiclib
Installing collected packages: geographiclib, geopy
Successfully installed geographiclib-1.49 geopy-1.20.0
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.5.11
  latest version: 4.7.10

Please update conda by running

    $ conda update -n base -c defaults conda



# All requested packages already installed.

Libraries imported.
```

[3]:
```
url = 'https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_M'
toronto_list= pd.read_html(url)[0]
```

```
        ---------------------------------------------------------------------------

        ImportError                               Traceback (most recent call last)

        <ipython-input-3-f17a33f5ba49> in <module>
          1 url = 'https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_M'
    ----> 2 toronto_list= pd.read_html(url)[0]


        ~/conda/envs/python/lib/python3.6/site-packages/pandas/io/html.py in read_html(io,␣
  ↪match, flavor, header, index_col, skiprows, attrs, parse_dates, thousands, encoding, decimal,␣
  ↪converters, na_values, keep_default_na, displayed_only)
       1103         na_values=na_values,
       1104         keep_default_na=keep_default_na,
    -> 1105         displayed_only=displayed_only,
       1106     )


        ~/conda/envs/python/lib/python3.6/site-packages/pandas/io/html.py in _parse(flavor, io,␣
  ↪match, attrs, encoding, displayed_only, **kwargs)
        886     retained = None
        887     for flav in flavor:
```

```
--> 888         parser = _parser_dispatch(flav)
    889         p = parser(io, compiled_match, attrs, encoding, displayed_only)
    890


    ~/conda/envs/python/lib/python3.6/site-packages/pandas/io/html.py in _parser_dispatch(flavor)
    841     else:
    842         if not _HAS_LXML:
--> 843             raise ImportError("lxml not found, please install it")
    844     return _valid_parsers[flavor]
    845


    ImportError: lxml not found, please install it
```

```
[ ]: toronto_list.head()
```

```
[ ]:
```

```
[ ]: CanadaData = pd.DataFrame(toronto_list)
```

```
[ ]: CanadaData.shape
```

```
[ ]: CanadaData.head()
```

```
[ ]: CanadaData = CanadaData[CanadaData.Borough != "Not assigned"]
```

```
[ ]: CanadaData.head()
```

```
[ ]: CanadaData.shape
```

```
[ ]: CanadaData.loc[CanadaData.Neighbourhood == 'Not assigned', 'Neighbourhood'] = CanadaData.Borough
```

```
[ ]: CanadaDataGrouped = CanadaData.groupby(['Postcode', 'Borough'], as_index=False, sort=False).agg(','.join)
```

```
[ ]: CanadaDataGrouped.head()
```

```
[ ]: LatitudeLongitudeData = pd.read_csv("http://cocl.us/Geospatial_data")
```

```
[ ]: LatitudeLongitudeData.head()
```

```
[ ]: LatitudeLongitudeData.rename(columns={'Postal Code':'Postcode'}, inplace=True)
```

```
[ ]: LatitudeLongitudeData.head()
```

```
[ ]: CanadaDataMerged = pd.merge(CanadaDataGrouped, LatitudeLongitudeData, on='Postcode')
```

```
[ ]: CanadaDataMerged.head()
```

```
[ ]: toronto_data = CanadaDataMerged[CanadaDataMerged['Borough'].str.contains("Toronto")].reset_index(drop=True)
```

[ ]:
```
toronto_data.shape
```

[ ]:
```
address = 'Toronto, Canada'

geolocator = Nominatim(user_agent="Canada_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Toronto are {}, {}.'.format(latitude, longitude))
```

[ ]:
```
map_Toronto = folium.Map(location=[latitude, longitude], zoom_start=11)

# add markers to map
for lat, lng, label in zip(toronto_data['Latitude'], toronto_data['Longitude'], toronto_data['Neighbourhood']):
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_Toronto)

map_Toronto
```

[ ]:
```
%%html
<img src="Folium_map1.jpg",width="200",height="200">
```

[ ]:
```
CLIENT_ID = 'L2YC5V3P20VYR5G54RNBYOGX5KHVPCESQJNCWOKDENXKZOKS' # your Foursquare ID
CLIENT_SECRET = 'MZSNJU4D3JBWXJWPARX0XU1PQG0DI50L3SAGURYJP2HJXPES' # your Foursquare Secret
VERSION = '20180605' # Foursquare API version

print('Your credentails:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)
```

[ ]:
```
toronto_data.loc[0, 'Neighbourhood']
```

[ ]:
```
neighborhood_latitude = toronto_data.loc[0, 'Latitude'] # neighborhood latitude value
neighborhood_longitude = toronto_data.loc[0, 'Longitude'] # neighborhood longitude value

neighborhood_name = toronto_data.loc[0, 'Neighbourhood'] # neighborhood name
```

```
print('Latitude and longitude values of {} are {}, {}.'.format(neighborhood_name,
                                                               neighborhood_latitude,
                                                               neighborhood_longitude))
```

[ ]:

```
LIMIT = 100 # limit of number of venues returned by Foursquare API

radius = 500 # define radius

url = 'https://api.foursquare.com/v2/venues/explore?
 ↪&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    neighborhood_latitude,
    neighborhood_longitude,
    radius,
    LIMIT)
url
```

[ ]:

```
results = requests.get(url).json()
results
```

[ ]:

```
def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']
```

[ ]:

```
venues = results['response']['groups'][0]['items']

nearby_venues = json_normalize(venues) # flatten JSON

# filter columns
filtered_columns = ['venue.name', 'venue.categories', 'venue.location.lat', 'venue.location.lng']
nearby_venues =nearby_venues.loc[:, filtered_columns]

# filter the category for each row
nearby_venues['venue.categories'] = nearby_venues.apply(get_category_type, axis=1)

# clean columns
nearby_venues.columns = [col.split(".")[-1] for col in nearby_venues.columns]

nearby_venues.head()
```

[ ]:

```
print('{} venues were returned by Foursquare.'.format(nearby_venues.shape[0]))
```

[ ]:

```
def getNearbyVenues(names, latitudes, longitudes, radius=500):

    venues_list=[]
    for name, lat, lng in zip(names, latitudes, longitudes):
        print(name)

        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?
  ↪&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
            CLIENT_ID,
            CLIENT_SECRET,
            VERSION,
            lat,
            lng,
            radius,
            LIMIT)

        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['items']

        # return only relevant information for each nearby venue
        venues_list.append([(
            name,
            lat,
            lng,
            v['venue']['name'],
            v['venue']['location']['lat'],
            v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])

    nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_list])
    nearby_venues.columns = ['Neighborhood',
                  'Neighborhood Latitude',
                'Neighborhood Longitude',
                  'Venue',
                  'Venue Latitude',
                  'Venue Longitude',
                  'Venue Category']

    return(nearby_venues)
```

[ ]:

```
toronto_venues = getNearbyVenues(names=toronto_data['Neighbourhood'],
                                   latitudes=toronto_data['Latitude'],
                                   longitudes=toronto_data['Longitude']
                                  )
```

[ ]:
```
print(toronto_venues.shape)
toronto_venues.head()
```

[ ]:
```
toronto_venues.groupby('Neighborhood').count()
```

[ ]:
```
print('There are {} uniques categories.'.format(len(toronto_venues['Venue Category'].unique())))
```

[ ]:
```
toronto_onehot = pd.get_dummies(toronto_venues[['Venue Category']], prefix="",␣
 ↪prefix_sep="")

# add neighborhood column back to dataframe
toronto_onehot['Neighborhood'] = toronto_venues['Neighborhood']

# move neighborhood column to the first column
fixed_columns = [toronto_onehot.columns[-1]] + list(toronto_onehot.columns[:-1])
toronto_onehot = toronto_onehot[fixed_columns]

toronto_onehot.head()
```

[ ]:
```
toronto_onehot.shape
```

[ ]:
```
toronto_grouped = toronto_onehot.groupby('Neighborhood').mean().reset_index()
toronto_grouped
```

[ ]:
```
toronto_grouped.shape
```

[ ]:
```
num_top_venues = 5

for hood in toronto_grouped['Neighborhood']:
    print("----"+hood+"----")
    temp = toronto_grouped[toronto_grouped['Neighborhood'] == hood].T.reset_index()
    temp.columns = ['venue','freq']
    temp = temp.iloc[1:]
    temp['freq'] = temp['freq'].astype(float)
    temp = temp.round({'freq': 2})
    print(temp.sort_values('freq', ascending=False).reset_index(drop=True).
 ↪head(num_top_venues))
    print('\n')
```

[ ]:
```
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)

    return row_categories_sorted.index.values[0:num_top_venues]
```

[ ]:
```
num_top_venues = 10

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['Neighborhood']
```

```
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = toronto_grouped['Neighborhood']

for ind in np.arange(toronto_grouped.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] =␣
  ↪return_most_common_venues(toronto_grouped.iloc[ind, :], num_top_venues)

neighborhoods_venues_sorted.head()
```

[ ]:
```
kclusters = 5

toronto_grouped_clustering = toronto_grouped.drop('Neighborhood', 1)

# run k-means clustering
kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(toronto_grouped_clustering)

# check cluster labels generated for each row in the dataframe
kmeans.labels_[0:10]
```

[ ]:
```
neighborhoods_venues_sorted.head()
```

[ ]:
```
neighborhoods_venues_sorted.rename(columns={'Neighborhood':'Neighbourhood'},␣
  ↪inplace=True)
```

[ ]:
```
neighborhoods_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)

toronto_merged = toronto_data

# merge toronto_grouped with toronto_data to add latitude/longitude for each neighborhood
toronto_merged = toronto_merged.join(neighborhoods_venues_sorted.
  ↪set_index('Neighbourhood'), on='Neighbourhood')

toronto_merged.head()
```

[ ]:
```
map_clusters = folium.Map(location=[latitude, longitude], zoom_start=11)

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]
```

```
# add markers to the map
markers_colors = []
for lat, lon, poi, cluster in zip(toronto_merged['Latitude'], toronto_merged['Longitude'],␣
  ↪toronto_merged['Neighbourhood'], toronto_merged['Cluster Labels']):
    label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse_html=True)
    folium.CircleMarker(
        [lat, lon],
        radius=5,
        popup=label,
        color=rainbow[cluster-1],
        fill=True,
        fill_color=rainbow[cluster-1],
        fill_opacity=0.7).add_to(map_clusters)

map_clusters
```

[ ]:

```
%%html
<img src="Folium_map2.jpg",width="200",height="200">
```

[ ]:

```
toronto_merged.loc[toronto_merged['Cluster Labels'] == 0, toronto_merged.columns[[1] +␣
  ↪list(range(5, toronto_merged.shape[1]))]]
```

[ ]:

```
toronto_merged.loc[toronto_merged['Cluster Labels'] == 1, toronto_merged.columns[[1] +␣
  ↪list(range(5, toronto_merged.shape[1]))]]
```

[ ]:

```
toronto_merged.loc[toronto_merged['Cluster Labels'] == 2, toronto_merged.columns[[1] +␣
  ↪list(range(5, toronto_merged.shape[1]))]]
```

[ ]:

```
toronto_merged.loc[toronto_merged['Cluster Labels'] == 3, toronto_merged.columns[[1] +␣
  ↪list(range(5, toronto_merged.shape[1]))]]
```

[ ]:

```
toronto_merged.loc[toronto_merged['Cluster Labels'] == 4, toronto_merged.columns[[1] +␣
  ↪list(range(5, toronto_merged.shape[1]))]]
```