coords

November 4, 2019

[1]:
```
import pandas as pd
import folium
import folium.plugins
import warnings
import numpy as np
warnings.filterwarnings('ignore')
from matplotlib import pyplot as plt
from matplotlib.pyplot import rc
```

1 WA Crash Viz and Analysis

1.0.1 By Katharine Chen, Tianqi Fang, Yutong Liu, Shuyi Yin

1.1 Problem Background

- A broad variety of factors (environmental, physical, etc) contribute to a road's overall safety;
- Professional and non-professional users all need user-friendly interfaces to efficiently query and analyze data;

1.2 Our solution

Ideally, we want to develop a **website** that: + let all users select filters of **time, weather, road, vehicle, pedestrian, etc**; + let average drivers view past accidents on their planned routes and pop-up info; + let semi-professional users obtain reports on selected data and view factor contribution bar plots; + let professional users (who can code) have the option to further connect with and develop on the platform;

1.3 Data Sets

Highway Safety Information System (HSIS)

1.4 Use case

- Average driver may consult the map and analysis report before travel;

- DOT planners, police officers and other professionals may look deeper into contributing factors;
- They are all **non-programmers** and thus need an interactive environment that visualize past accidents and tell them what factors contribute most to crashes;
- ***Rainy day, steep downhill curved road, old car, little traffic, young driver;***

1.5 Python Libraries:

- Folium;
- Bokeh;

1.6 Data cleaning

[ ]:
```
def clean(source, sink):
    df = pd.read_csv(source)
    df = df.dropna()
    df.to_csv(sink)
    pass
```

[ ]:
```
# use case:
clean("coords.csv", "coords_cleaned.csv")
```

1.6.1 Load and wrangle dataset with Pandas

[2]:
```
def readCleanLoc(source):
    '''
    @param source: cleaned csv file of lat/lons
    @use case: df_cleaned = readLoc("coords_gps.csv")
    '''
    # read from file
    df_cleaned = pd.read_csv(source)

    # change column name
    df_cleaned.columns = ['lat', 'lon']

    # assert no nan after cleaning
    assert not df_cleaned.isnull().values.any()
    return df_cleaned
```

[3]:
```
def readOrig(origSource, cleanLocSource):
    '''
    @param origSource: original dataset file
    @param cleanLocSource: cleaned lat/lon file
    @use case: df = readOrig("WA_Rural_St_RtesCrashes_Full.csv")
```

```
    '''

    # read
    df = pd.read_csv(origSource)

    # select columns
    columns = ['DATE',
               'PRIMARY TRAFFICWAY',
               'MILEPOST',
               '# INJ',
               '# FAT',
               'WA STATE PLANE SOUTH - X 2010 - FORWARD',
               'WA STATE PLANE SOUTH - Y 2010 - FORWARD']
    df = df[columns]

    # drop rows that have nan in defined columns
    df = df.dropna(subset=['WA STATE PLANE SOUTH - X 2010 - FORWARD',
                           'WA STATE PLANE SOUTH - Y 2010 - FORWARD'])

    # must collapse index
    df = df.reset_index(drop=True)

    # transform time
    df['date'] = pd.to_datetime(df['DATE'])
    df['year'] = df['date'].dt.year

    columns = ['PRIMARY TRAFFICWAY',
               'MILEPOST',
               '# INJ',
               '# FAT',
               'WA STATE PLANE SOUTH - X 2010 - FORWARD',
               'WA STATE PLANE SOUTH - Y 2010 - FORWARD',
               'year']
    df = df[columns]

    # change columns name
    df.columns = ['PRIMARY TRAFFICWAY',
                  'MILEPOST',
                  '# INJ',
                  '# FAT',
                  'lat',
                  'lon',
                  'year']

    # update lat lon
    df_cleaned = readCleanLoc(cleanLocSource)
    df.lat = df_cleaned.lat
```

```
    df.lon = df_cleaned.lon

    # assert no nan
    assert not df.isnull().values.any()
    assert not df_cleaned.isnull().values.any()

    return df
```

1.7 Folium simple example

[4]:
```
def plotFolium(origFile, cleanLoc, mapSink, start, end):
    '''
    @param origFile: original dataset file
    @param cleanLoc: cleaned lat/lon file
    @param mapSink: saving destination of generated map
    '''

    # read data
    df = readOrig(origFile, cleanLoc)

    # create map object
    accWA = folium.Map([df.lat.median(), df.lon.median()],
                   # tiles="cartodbpositron",
                   tiles = '',
                   # width='80%',
                   # height='80%',
                   prefer_canvas=True,
                   zoom_start=8)
    # add tile layer
    folium.TileLayer('cartodbpositron', name = 'bright').add_to(accWA)
    folium.TileLayer('CartoDB dark_matter', name = 'dark').add_to(accWA)

    # create crash layer
    crashes = []
    clusters = []
    for year in range(start, end + 1):

        # create cluster layer
        yrClust = folium.FeatureGroup(name=str(year) + '_Clusters', show=False)
        clusters.append(yrClust)
        accWA.add_child(clusters[-1])

        # add cluster layer to feature group
        marClst = folium.plugins.FastMarkerCluster(
            data=list(zip(df[df['year'] == year]['lat'].values, df[df['year'] ==␣
↪year]['lon'].values))
```

```
        ).add_to(clusters[-1])

        # individual crashes
        yrCrash = folium.FeatureGroup(name=str(year) + '_Crashes', show=False)
        crashes.append(yrCrash)
        accWA.add_child(crashes[-1])

        # add crashe events to their layers
        for _, row in df[df['year'] == year].iterrows():

            # define circle color
            if row['# INJ'] > 0:
                cirlColor = "#007849"
            elif row['# FAT'] > 0:
                cirlColor = 'red'
            else:
                cirlColor = 'steelblue'

            # define circle radius
            if row['# INJ'] + row['# FAT'] > 0:
                cirlRadius = max(row['# INJ'], row['# FAT']) * 3
            else:
                cirlRadius = 1

            folium.CircleMarker([row['lat'], row['lon']],
                                radius=cirlRadius,
                                popup=folium.Popup("INJ: {}, FAT: {}".format(
                                row['# INJ'], row['# FAT']), max_width=150),
                                # fill_color="#3db7e4",
                                # color=cirlColor,
                                weight = 0.2,
                                fill_color=cirlColor,
                                fill=True,
                                fill_opacity=0.4
                            ).add_to(crashes[-1])



    # add layer control
    folium.LayerControl().add_to(accWA)

    # save map
    accWA.save(mapSink)

    return accWA
```

Do not excecute the following cell

[ ]:
```
# add layers of crashes by year
for year in ...:
    yrCrash = folium.FeatureGroup(name=str(year) + '_Crashes', show=False)
    crashes.append(yrCrash)
    accWA.add_child(crashes[-1])
```

[ ]:
```
# plot the individual events
for year in ...:

    # iterate in a loop
    for _, row in df[df['year'] == year].iterrows():

        ...

        folium.CircleMarker([row['lat'], row['lon']],
                            radius=cirlRadius,
                            popup=folium.Popup("INJ: {}, FAT: {}".format(
                            row['# INJ'], row['# FAT']), max_width=150),
                            weight = 0.2,
                            fill_color=cirlColor,
                            fill=True,
                            fill_opacity=0.4
                        ).add_to(crashes[-1])
        # events are added to its corresponding layer by Year
```

[5]:
```
_ = plotFolium("WA_Rural_St_RtesCrashes_Full.csv",
            "coords_gps.csv",
            "folium_year.html",
            2014, 2017)
```

View the Crashes By Year plot here;

View the Crashes Injuries By County plot here;

View the Crashes Fatalities By County plot here;

1.8 Bokeh simple example

[6]:
```
def plotBokeh(origFile, cleanLoc, mapSink, start, end):

    # import local modules
    import math
    from bokeh.io import show
    from bokeh.palettes import brewer
```

```
from bokeh.models import ColumnDataSource
from bokeh.plotting import figure, output_file, save
from bokeh.tile_providers import get_provider, Vendors

# def coordinate conversion
def merc(coords):
    lat = coords[0]
    lon = coords[1]

    r_major = 6378137.000
    x = r_major * math.radians(lon)
    scale = x/lon
    y = 180.0/math.pi * math.log(math.tan(math.pi/4.0 +
        lat * (math.pi/180.0)/2.0)) * scale
    return (x, y)



# read data
df = readOrig(origFile, cleanLoc)

for year in range(start, end + 1):
    output_file("bokeh_year_{}.html".format(year))
    tile_provider = get_provider(Vendors.CARTODBPOSITRON)

    cirlColor = []
    cirlRadius = []
    for _, row in df[df['year'] == year].iterrows():
        # define circle color
        if row['# INJ'] > 0:
            cirlColor.append("#007849" )
        elif row['# FAT'] > 0:
            cirlColor.append('red')
        else:
            cirlColor.append('steelblue')

        # define circle radius
        if row['# INJ'] + row['# FAT'] > 0:
            cirlRadius.append(max(row['# INJ'], row['# FAT']) * 3)
        else:
            cirlRadius.append(1)


    # range bounds supplied in web mercator coordinates
    p = figure(#x_range=(-14000000, -12800000), y_range=(5900000, 6100000),
               x_axis_type="mercator", y_axis_type="mercator",
```

```
                    plot_width = 1600, plot_height = 1200, title = '{} crashes'.
↪format(year))
        p.add_tile(tile_provider)
        p.sizing_mode = 'stretch_both'

        z = map(merc, df[df['year'] == year][['lat','lon']].values)
        z = list(z)

        coords_x = [x[0] for x in z]
        coords_y = [x[1] for x in z]

        # p = figure(plot_width=400, plot_height=400)
        p.circle(x=coords_x, y=coords_y, color=cirlColor, fill_alpha=0.8,␣
↪size=cirlRadius)

        p.yaxis.axis_label = "Latitude"
        p.yaxis.axis_label_text_font_size = '20pt'
        p.yaxis.major_label_text_font_size = '20pt'

        p.xaxis.axis_label = "Longitude"
        p.xaxis.axis_label_text_font_size = '20pt'
        p.xaxis.major_label_text_font_size = '20pt'
        # p.yaxis.axis_label_text_font = "times"
        # p.yaxis.axis_label_text_color = "black"
        save(p)

    pass
```

Do not execute the following cell

[7]:
```
# convert lat/lon to mercator coordinates
def merc(coords):
    lat = coords[0]
    lon = coords[1]

    r_major = 6378137.000
    x = r_major * math.radians(lon)
    scale = x/lon
    y = 180.0/math.pi * math.log(math.tan(math.pi/4.0 +
        lat * (math.pi/180.0)/2.0)) * scale
    return (x, y)
```

[ ]:
```
# here we are not creating layers, but HTML files
for year in ...:
    p = figure(x_axis_type="mercator", y_axis_type="mercator",
               plot_width = 1600, plot_height = 1200, title = str(year))
    p.add_tile(tile_provider)
```

```
    z = map(merc, df[df['year'] == year][['lat','lon']].values)
    z = list(z)

    coords_x = [x[0] for x in z]
    coords_y = [x[1] for x in z]

    # plot in batch, instead of in loops
    p.circle(x=coords_x, y=coords_y, color=cirlColor, fill_alpha=0.8,␣
↪size=cirlRadius)
```

[8]:
```
plotBokeh("WA_Rural_St_RtesCrashes_Full.csv",
            "coords_gps.csv",
            "folium_year.html",
            2014, 2017)
```

View the Crashes of Year 2016 plot here;

View the Crashes of Year 2017 plot here;

1.9 Comparison of Folium vs. Bokeh vs. Matplotlib

[ ]:
```
%%html
<style>
table {float:left}
</style>
```

Folium	Bokeh	Matplotlib
Interactive	Interactive	Static
Zoom in/out easilywith scroll	Need to select zoom mode	N/A
Allow multiple features easily:layers and clusters	May be possible,but hard	N/A
Slow	Fast	N/A

[ ]:
```
fig, ax = plt.subplots(figsize=(8,6))

width = 0.35
labels = ['2014-2017', '2017']
multi = [77.6, 8.13]
one = [20.8, 2.90]

ax.bar(np.arange(2) - width/2, multi, width, label='Folium')
ax.bar(np.arange(2) + width/2, one, width, label='Bokeh')
```

```
ax.set_ylabel('Plotting time (s)', fontsize=14)
ax.set_title('Execution time for Folium and Bokeh to plot\n 1 and 4 years of␣
 ↪data', fontsize=14)
ax.set_xticks(np.arange(2))
ax.set_xticklabels(labels, fontsize=14)
ax.set_xlim(-1,2)

ax.legend(fontsize=14)
plt.savefig('times.png')
```

1.10 Thanks! Q&A time

[ ]:

1.11 Aggregate by year

[ ]:
```
df = pd.read_csv("WA_Rural_St_RtesCrashes_Full.csv")
df['date'] = pd.to_datetime(df['DATE'])
```

[ ]:
```
df['year'] = df['date'].dt.year
df.groupby(['year']).agg(['count'])
```

1.11.1 Num of cases across years

[ ]:
```
df['year'].value_counts()
```

[ ]:
```
from matplotlib import pyplot as plt
plt.plot(df['year'].value_counts().sort_index())
plt.show()
```