session2_data_viz_demo

September 20, 2019

1 Data Visualization with Python

Kim Hee (Graduate research assistant) Universitätsmedizin Mannheim, Mannheim (UMM)

- This is prepared for `Data analysis tools (Datenanalysewerkzeuge)` at MIRACUM summer school 2019

Agenda * Case Study 1: Dr. John Snow Cholera Outbreak (Geodata Visualization) * Case Study 2: Labitems Trend Visualization

1.1 Case Study 1: Dr. John Snow Cholera Outbreak (Geodata Visualization)

This case study will recreate the approach that Dr. John Snow solved the cholera outbreak in London in 1854.

Dr. John Snow was a physician from London in 1854 when a cholera outbreak occurred in the Soho district, in the West End of London. There was widespread belief that "Miasma" or bad air and stench was the cause of the cholera outbreak in Soho district. However, Dr. Snow was skeptical of the Miasma theory and was certain that cholera was likely a water-borne illness. In addition to his systematic process of determining how cholera is transmitted, he began to create maps of the Cholera deaths in Soho and to map the locations of water pumps within the neighborhood. Original map by Dr. John Snow shown on the right showing the clusters of cholera cases in the London epidemic of 1854, drawn and lithographed by Charles Cheffins.

1.1.1 Import Folium (1/5)

https://python-visualization.github.io/folium/# `folium` is a Python web mapping package that can map data on a Leaflet map.

[1]:
```
import folium
from folium import plugins
import pandas as pd
```

1.1.2 Create a basemap of Soho District (2/5)

`Foluim` will plot the map object based on `OpenStreetMap (OSM)`. `OSM` is a publicly available world map based on crowdsourced geographic information.

[10]:
```
# the latitude and Longitude coordinates of the Soho District center
SOHO_COORDINATES = (51.513578, -0.136722)
map_soho = folium.Map(location = SOHO_COORDINATES, width = "100%", zoom_start =␣
↪15) # max zoom: 18
map_soho
```

[10]:
```
<folium.folium.Map at 0x7fcf61ea8828>
```

Plot the map with Stamentoner themes that looks more similar to the original cholera map FYI, another popular theme is 'cartodbpositron'

[11]:
```
folium.TileLayer('stamentoner').add_to(map_soho)
map_soho
```

[11]:
```
<folium.folium.Map at 0x7fcf61ea8828>
```

1.1.3 Load and prepare the data (3/5)

The mortality data set has been collected by Dr. Snow from the Registrar's Office and from hospital records.

[4]:
```
df_pumps = pd.read_csv('data/johnsnow_pumps.csv')
df_pumps.head(3)
```

[4]:

	FID	LON	LAT
0	250	-0.136668	51.513341
1	251	-0.139586	51.513876
2	252	-0.139671	51.514906

[5]:
```
df_deaths = pd.read_csv('data/cholera_deaths.csv')
df_deaths.head(3)
```

[5]:

	FID	DEATHS	LON	LAT
0	0	3	-0.137930	51.513418
1	1	2	-0.137883	51.513361
2	2	1	-0.137853	51.513317

Based on the given data set, three list objects need to be created. * `coordinates_p` stores all coordinates of pumps * `coordinates_d` stores all coordinates of deaths * `totaldeath` contains the number of deaths for each coordinate. This will determine the radius size of a marker

[6]:
```
coordinates_p = df_pumps[["LAT","LON"]].values.tolist()
coordinates_d = df_deaths[["LAT","LON"]].values.tolist()
totaldeaths   = df_deaths[["DEATHS"]].values.tolist()
# Note that multiple functions can be chained together like above: .values.
↪to_list()
```

1.1.4 Mapping the mortality data to the basemap (4/5)

Let us augument the basemap with mortality data set. Iterate the data set and map the `LAT` and `LON` values in `coordinates_d`. RegularPolygonMarker creates custom markers that draw red circles instead of points. The radius of the circle will be determined by the number of deaths in `totaldeaths`

[12]:
```
for i in range(0, len(coordinates_d)):
    folium.RegularPolygonMarker(coordinates_d[i], radius = totaldeaths[i], \
                                stroke = False, fill_color = "red",␣
 ↪fill_opacity = 0.5, number_of_sides = 12
                               ).add_to(map_soho)
map_soho
```

[12]:
```
<folium.folium.Map at 0x7fcf61ea8828>
```

1.1.5 Mapping the water pump data to the basemap (5/5)

[13]:
```
for i in range(0, len(coordinates_p)):
    folium.RegularPolygonMarker(coordinates_p[i], radius = 10, \
                                stroke = False, fill_color = "blue",␣
 ↪fill_opacity = 1
                               ).add_to(map_soho)
map_soho
```

[13]:
```
<folium.folium.Map at 0x7fcf61ea8828>
```

The pump handle was removed on September 8, 1854 and remains as John Snow memorial on Broadwick Street, Soho. The public house named after Dr. John Snow is also seen behind the pump

Agenda * Case Study 1: Dr. John Snow Cholera Outbreak (Geodata Visualization) * Case Study 2: Labitems Trend Visualization

1.1.6 Interactive Web application with Dash

- https://dash.plot.ly/gallery
- Dash is a Python framework for creating data-driven web applications
- Dash apps are written on top of Flask, Plotly, and React
- Flask is a Python web framework
- Plotly is specifically a charting library built on top of D3.js
- React is a JavaScript library for building user interfaces maintained by Facebook and a community

1.2 Case Study 2: Labitems Trend Visualization

Trend analysis is the widespread practice of collecting information and attempting to spot a pattern. This case study will illustrate a drug reaction of a sepsis patient. This case study tracks the biomarker and prescription history of patient 41976. It visualizes the relation between two key biomarkers of sepsis (White Blood Cells and Neutrophils) and

- '41976' patient is choosen for this case study because this patient contains most and interesting records among other sepsis patients '10006', '10013', '10036', '10056', '40601'

1.2.1 Import Python pakages (1/6)

[10]:
```
import dash
import dash_core_components as dcc
import dash_html_components as html
import flask
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 999)
import pandas.io.sql as psql
import psycopg2
```

1.2.2 Data collection from database (2/6)

- Make a database connection
- Query `d_labitems` table (Dictionary table for mapping)
- Query `labevents` table (History of the labitem order)
- Join two tables
- Query `prescriptions` table (History of the prscription order)

[11]:
```
DB_IP = "129.206.5.27"
conn = psycopg2.connect(f"postgres://postgres:postgres@{DB_IP}:5432/mimic")

sql = "SELECT * FROM d_labitems"
d_lab = psql.read_sql(sql, conn)
d_lab.drop(columns = ['row_id'], inplace = True)

sql = "SELECT * FROM labevents WHERE subject_id IN (41976)"
lab = psql.read_sql(sql, conn)
lab.drop(columns = ['row_id'], inplace = True)

lab = pd.merge(d_lab, lab, on = 'itemid', how = 'inner')

sql = "SELECT * FROM prescriptions WHERE subject_id IN (41976)"
```

```
presc = psql.read_sql(sql, conn)
presc.drop(columns = ['row_id'], inplace = True)
```

1.2.3 Data preparation for labevents table (3/6)

- Convert data type to datetime and extract only year value

[12]:
```
lab['charttime'] = pd.to_datetime(lab['charttime'], errors = 'coerce')
lab.sort_values(by='charttime', inplace=True)
lab.set_index('charttime', inplace = True)
lab.head(1)
```

[12]:
```
                     itemid    label  fluid    category loinc_code  \
charttime
2198-09-29 07:50:00   51237  INR(PT)  Blood  Hematology     5895-7

                     subject_id  hadm_id value  valuenum valueuom      flag
charttime
2198-09-29 07:50:00       41976      NaN   3.6       3.6     None  abnormal
```

1.2.4 Data preparation for prescriptions table (4/6)

- Filter conditions:
- unit: 'mg'
- antibiotics medicines: ('Vancomycin','Meropenem','Levofloxacin')
- Contruct a normalized dose column
- Convert data type to datetime and extract only year value

[13]:
```
presc['dose_val_rx'] = pd.to_numeric(presc['dose_val_rx'], errors = 'coerce')
presc = presc[presc['dose_unit_rx']=='mg']
presc = presc[presc['drug'].isin(['Vancomycin','Meropenem','Levofloxacin'])]

temp_df = pd.DataFrame()
for item in presc.drug.unique():
    temp = presc[presc['drug'].str.contains(item)]
    temp['norm_size'] = temp['dose_val_rx'] / temp['dose_val_rx'].max()
    temp_df = temp_df.append(temp)
presc = pd.merge(presc, temp_df, on=list(presc.columns))

presc['startdate'] = pd.to_datetime(presc['startdate'], errors = 'coerce')
presc.sort_values(by='startdate', inplace=True)
presc.set_index('startdate', inplace = True)
presc.head(1)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8:
SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[13]:
```
            subject_id  hadm_id  icustay_id    enddate drug_type        drug  \
startdate
2198-10-29       41976   125449    285272.0 2198-11-01      MAIN  Vancomycin

           drug_name_poe drug_name_generic formulary_drug_cd     gsn  \
startdate
2198-10-29          None              None            VANC1F  043952

                    ndc  prod_strength  dose_val_rx dose_unit_rx  \
startdate
2198-10-29  00338355248  1g Frozen Bag       1000.0           mg

           form_val_disp form_unit_disp route  norm_size
startdate
2198-10-29             1            BAG    IV        1.0
```

1.2.5 Create a structure and presentation of your web with HTML and CSS (5/6)

[16]:
```
list_patient = ['41976']
list_biomarker = ['White Blood Cells', 'Neutrophils']
list_drug = ['Vancomycin','Meropenem','Levofloxacin']

stylesheets = ['./resources/bWLwgP.css']
app = dash.Dash()

app.layout = html.Div([

    dcc.Dropdown(
        id = 'patient',
        value = '41976',
        multi = False,
        options = [{'label': i, 'value': i} for i in list_patient],
    ),
    dcc.Dropdown(
        id = 'biomarker',
        value = 'White Blood Cells',
        multi = False,
```

```
        options = [{'label': i, 'value': i} for i in list_biomarker],
    ),
    dcc.Dropdown(
        id = 'drug',
        value = ['Vancomycin'],
        multi = True,
        options = [{'label': i, 'value': i} for i in list_drug],
    ),
    dcc.Graph(id = 'graph'),
])
```

1.2.6 Define the reactive behavior with Python (6/6)

[17]:
```
@app.callback(Output('graph', 'figure'),
              [Input('patient', 'value'),
               Input('biomarker', 'value'),
               Input('drug', 'value')])
def update_graph(patient, biomarker, drug):
    traces = []
    temp_l = lab[lab['subject_id'].astype(str) == patient]
    temp_p = presc[presc['subject_id'].astype(str) == patient]
    temp_min = 0

    item = biomarker
    temp = temp_l[temp_l['label'] == item]
    temp_min = float(temp.value.astype(float).min())
    trace = go.Scatter(
                x = temp.index,
                y = temp.value,
                name = item,
                mode = 'lines+markers',
            )
    traces.append(trace)

    for i, item in enumerate(drug):
        temp = temp_p[ temp_p['drug'] == item]
        trace = go.Scatter(
                    x = temp.index,
                    y = np.ones((1, len(temp)))[0] * temp_min - i - 1,
                    name = item,
                    mode = 'markers',
                    marker = {
                        'size': temp.norm_size * 10
                    }
                )
        traces.append(trace)
```

```
    layout = go.Layout(
        legend = {'x': 0.5, 'y': -0.1, 'orientation': 'h', 'xanchor': 'center'},
        margin = {'l': 300, 'b': 10, 't': 10, 'r': 300},
        hovermode = 'closest',
    )
    return {'data': traces, 'layout': layout}

# NOTE that you can access the web app via http://IP_ADDRESS:PORT/
app.run_server(host = '0.0.0.0', port = 8063)
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8050/ (Press CTRL+C to quit)
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET / HTTP/1.1" 200 -
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_renderer/react@16.8.6.min.js?v=1.1.0&m=1568818297 HTTP/1.1" 200 -
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_renderer/prop-types@15.7.2.min.js?v=1.1.0&m=1568818297 HTTP/1.1" 200
-
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_renderer/react-dom@16.8.6.min.js?v=1.1.0&m=1568818297 HTTP/1.1" 200
-
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_core_components/highlight.pack.js?v=1.2.0&m=1568821458 HTTP/1.1" 200
-
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_html_components/dash_html_components.min.js?v=1.0.1&m=1566962730
HTTP/1.1" 200 -
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_renderer/dash_renderer.min.js?v=1.1.0&m=1568818297 HTTP/1.1" 200 -
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_core_components/dash_core_components.min.js?v=1.2.0&m=1568821458
HTTP/1.1" 200 -
129.206.173.131 - - [19/Sep/2019 18:18:36] "GET /_dash-component-
suites/dash_core_components/plotly-1.49.4.min.js?v=1.2.0&m=1568821458 HTTP/1.1"
200 -
129.206.173.131 - - [19/Sep/2019 18:18:37] "GET /_dash-layout HTTP/1.1" 200 -
129.206.173.131 - - [19/Sep/2019 18:18:37] "GET /_dash-dependencies HTTP/1.1"
200 -
129.206.173.131 - - [19/Sep/2019 18:18:37] "POST /_dash-update-component
HTTP/1.1" 200 -
129.206.173.131 - - [19/Sep/2019 18:18:40] "POST /_dash-update-component
```

```
HTTP/1.1" 200 -
```

2 Question?

Kim Hee (Graduate research assistant) Universitätsmedizin Mannheim, Mannheim (UMM)