1 Lecture 10b: Interactivity and a quick introduction to Dash library

Data Visualization · 1-DAV-105

Lecture by Broňa Brejová

[1]:
```
# importing libraries
import numpy as np
import pandas as pd
import plotly.express as px
```

```
/tmp/ipykernel_21487/1620516280.py:3: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better
interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

1.1 Interactivity

Interactive visualization engages audience, allows them to explore data in depth and according to their interest.

1.1.1 Examples

- US cities with the same name (website), see also the animated explainer
- PhD gender gap (website)
- Making it big (website), more animated than interactive

1.1.2 Techniques in interactivity visualization

Similar to decisions made in designing a static plot:

- Selecting variables (x, y, color, …)
- Filtering data (selecting table rows)
- Highlighting points or groups
- Aggregating (display countries or region summaries)
- Zooming / panning
- Rescaling (log-scale) / reexpressing (e.g. % instead of counts)
- Sorting (e.g. bars in bargraphs)
- Displaying details (tooltips)
- Annotating
- Bookmarking

(Stephen Few)

1.1.3 Dashboard

- A display consisting of multiple plots, summarizing the current state of important indicators (e.g. of a business, pandemics, …)
- Inspired by dashboards in cars and planes
- Often interactive, but main features in default view

Two SARS-CoV-2 examples:

- WHO
- Nextstrain
  - many options: selecting color, filtering, highlighting, aggregating, zooming and panning (maps and tree), rescaling (time vs divergence), tooltips, bookmarking

1.1.4 Interactivity in Plotly Express

All Plotly plots by default have some interactivity:

- Filtering groups
- Zooming / panning
- Details
- Spike lines

Example 1: switching off categories Country indicators from World Bank, https://databank.worldbank.org/home under CC BY 4.0 license. Regions can be switched on and off.

[2]:
```
url = 'https://fmfi-compbio.github.io/viz/data/World_bank.csv'
countries = pd.read_csv(url)

px.scatter(
        countries, x="GDP2020", y="Expectancy2020", color="Region",
        hover_data=['Country'],
        title="Country indicators 2020", log_x=True,
        width=800, height=500
    )
```

1.1.5 Example 2: Spike lines

Life expectancy data, based on free data from World bank via gapminder.org, CC-BY license (years 1900-2017) and World bank directly (years 2018-2021).

Compare data along a chosen year.

[3]:
```
url = "https://fmfi-compbio.github.io/viz/data/life_expectancy_years.csv"
orig_expectancy = pd.read_csv(url, index_col=0)
orig_expectancy = orig_expectancy.iloc[:, 1:].reset_index()
expectancy = pd.melt(orig_expectancy, id_vars=["Country"], var_name="Year",␣
 ↪value_name="Expectancy")
expectancy['Year'] = expectancy['Year'].astype(int)
display(expectancy.head())
```

```
                Country  Year  Expectancy
0           Afghanistan  1900        29.4
1               Albania  1900        35.4
2               Algeria  1900        30.2
3                Angola  1900        29.0
4  Antigua and Barbuda  1900        33.8
```

[4]:
```
selected = expectancy.query("Country=='Slovak Republic' or Country=='Portugal'")
fig=px.line(
        selected, x="Year", y="Expectancy", color="Country",
        width=800, height=500
    )
fig.update_layout(hovermode="x unified")
```

1.1.6 Next: More interaction with Dash by Plotly

- Dash library by Plotly allows adding control elements (selectors, sliders, buttons, …)

1.2 Data for Dash: a simple table of function values

Function `np.linspace` below creates a list of 5 evenly spaced numbers in interval $[1, 3]$, stored as an object of array class from the Numpy library.

[5]:
```
x = np.linspace(start=1, stop=3, num=5)
print('x:', x)
```

```
x: [1.  1.5 2.  2.5 3. ]
```

The function below gets values of x and a dictionary of named mathematical functions, each consisting of function values of each value of x and converts this into a long `DataFrame` with columns `x`, `value`, and `function`. Zeroes and negative values are masked as missing to avoid problems with logarithmic y axis.

[6]:
```
def convert_table(x, function_dict):
    """ x is a list (or Numpy array) of values of x,
    function_dict is a dictionary containing function names as keys
    and lists of function values as values. The result will be a Pandas
    DataFrame (table) with each row containing triple x, function, value.
    Zeroes and negative values are masked as missing
    to avoid problems with logarithmic y axis."""

    # check that all functions have the same number of values as x
    for f in function_dict:
        assert(len(function_dict[f])==len(x))

    # create a wide table with each function as one column
    functions_wide = pd.DataFrame(function_dict, index=x)
    # reformat to long format
    #   where each row is a triple x, function name, function value
```

```
    functions = (functions_wide.reset_index()
                 .melt(id_vars='index')
                 .rename(columns={'variable':'function', 'index':'x'}))
    # mask values <= 0 as missing values
    val = functions['value']
    functions['value'] = val.mask(val <= 0, np.nan)
    return functions
```

[7]:
```
functions = convert_table(x, {'quadratic': x * x,'cubic': x * x * x})
```

Let us look at the resulting table `functions`:

- It has three columns named `'x'`, `'function'` and `'value'`.
- Each row is a triple, containing a function name and the values of x and $f(x)$.
- E.g. one of the rows for the cubic function has $x = 2$ and $f(x) = 2^3 = 8$.

[8]:
```
display(functions)
```

```
     x   function   value
0  1.0  quadratic   1.000
1  1.5  quadratic   2.250
2  2.0  quadratic   4.000
3  2.5  quadratic   6.250
4  3.0  quadratic   9.000
5  1.0      cubic   1.000
6  1.5      cubic   3.375
7  2.0      cubic   8.000
8  2.5      cubic  15.625
9  3.0      cubic  27.000
```

This DataFrame can be easily plotted using the Plotly library.

[9]:
```
figure = px.line(functions, x="x", y="value", color='function')
figure.show()
```

1.3 Interactive plots in Plotly Dash

- Dash library by Plotly allows adding control elements (selectors, sliders, buttons, …).
- It is not preinstalled in Colab, so the next line will install it.

[10]:
```
# if needed, uncommment this command to install the required version of Dash␣
 ↪library
# ! pip install dash==3.2.0
```

- The code below creates an interactive plot in which the user can choose which functions from the list to display.
- The code has many comments so read through it carefully.

[11]:
```
from dash import Dash, dcc, html, Input, Output
```

```
# create a list of all functions
all_functions = list(functions['function'].unique())

# create a new dash application app
app = Dash(__name__)

# Create layout of items in application
#   one html <div> item containing text as small headers (H4),
#   items for individual inputs and a graph at the bottom
# Currently we have two inputs:
#   an input field for entering title text
#   checkboxes for selecting functions
# These elements have identifiers which will be used later in the code
app.layout = html.Div([
    html.H4("Plot title: "),
    # input field for entering title text:
    dcc.Input(
        id='graph-title',
        type='text',
        value='My plot' # initial value
    ),
    html.H4("Select functions: "),
    # checkboxes for selecting functions:
    dcc.Checklist(
        id='selected-functions',
        options=all_functions,
        value=['quadratic'],  # initial value, quadratic function is selected
        inline=True   # place checkboxes horizontally
    ),
    # graph itself
    dcc.Graph(id='graph-content')
])

# @app.callback is a function decorator applied to function update_figure below.
# It defines that this function will be called to update the graph
#      when the user makes a change.
#    Input will be the value entered to the input field with id graph-title and
#      the list of functions selected in dcc.Checklist object with id␣
 ↪'selected-functions'.
#    Output will be the graph created by the function update_figure below,
#      which will be used to update dcc.Graph object with id 'graph-content'
@app.callback(
    Output('graph-content', 'figure'),
    [Input('graph-title', 'value'),
     Input('selected-functions', 'value')
    ]
)
```

```
def update_figure(title, selected_functions):
    """ Function for ploting the functions listed in list selected_functions
    with plot title given in title"""

    # select a subset of functions table with just those functions in input list
    functions_subset = functions.query('function in @selected_functions')

    # create a plotly line plot using the smaller table in functions_subset
    figure = px.line(
        functions_subset,
        x="x", y="value", color="function",
        width=800, height=500
    )

    # add title to the plot
    figure.update_layout(title_text=title)

    return figure

# run the whole application
app.run()
pass
```

```
<IPython.lib.display.IFrame at 0x731f95f58560>
```