1 Lecture 1c: Quick Introduction to New Libraries

Data Visualization · 1-DAV-105

Lecture by Broňa Brejová

We will now briefly discuss several libraries which will be used in the next tutorial. We will cover details of these libraries in later lectures, this is just a glipse of things to come.

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

1.1 Dangers of notebooks

- Frequent use pattern: the users have several cells finished and executed, they work on the last cell in the notebook, and run it repeatedly until it works correctly. This avoids repeated execution of the top cells, which may be slow.
- Notebooks do not force you to run cells in order from top to bottom. This generates problems if you skip some cells or execute them repeatedly.

Good practice suggestions:

- Avoid running cells out of order and occasionally restart the kernel (to remove variables) and run all cells (using a menu function).
- Do not modify variables introduced in other cells.
- Refactor bigger or repeated parts of code to functions. This also hides local variables from the rest of the notebook and thus prevents clashes.
- Ideally move functions to separate modules but this is harder in Colab.

Below we see an example not obeying the first recommendation; the value printed will depened on how many times we execute the second cell. This can lead to hard-to-find errors in a more complex case.

[2]:
```
value = 0
```

[3]:
```
value += 1
print(value)
```

```
1
```

See also I don't like notebooks by Joel Grus, an entertaining video explaining some of the pitfalls of notebooks

1.2 Libraries NumPy and Pandas

- Pandas is a Python library for working with tabular data.
- NumPy is a library of efficient multi-dimensional arrays used for numerical computations.

1.2.1 NumPy array and arithmetical operations with arrays

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

[4]:
```
import numpy as np
x = np.linspace(start=1, stop=3, num=5)
print('x:', x)
```

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

- We can do various arithmetic operations on whole NumPy arrays or apply predefined functions such as `np.exp`.
- Such operations are typically done element-by-element.

[5]:
```
print('x:', x)
print('x+1:', x+1)
print('x*x:', x*x)
print('np.exp(x):', np.exp(x))
```

```
x: [1.  1.5 2.  2.5 3. ]
x+1: [2.  2.5 3.  3.5 4. ]
x*x: [1.   2.25 4.   6.25 9.  ]
np.exp(x): [ 2.71828183  4.48168907  7.3890561  12.18249396 20.08553692]
```

1.2.2 Creating Pandas DataFrame

Below we create an object of Pandas DataFrame class, which is a 2-dimensional table. We will cover most Pandas functions used below next week, for now the details are not important.

[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
```

1.3 Displaying Pandas DataFrame using Plotly library

- Plotly is a popular library for plotting.
- It provides some advanced plot types and all plots are interactive.
- For example, we can also zoom into parts of the plot by selecting a rectangle.
- A menu with additional options appears in the top right corner of the plot.
- It is very convenient for displaying tables.
- In the `px.line` function we first give the table `functions` and then specify, which columns should be used as x coordinate, y coordinate and color.
- One line will be automatically drawn for each distinct value in the `color` column and a legend will be added.

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

1.4 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.

[ ]:
```
! pip install dash
```

- 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_server(mode='inline')
```

```
<IPython.lib.display.IFrame at 0x7f01ed72d630>
```