Lab 03 -

COMP7507 Visualization and Visual Analytics

2024 Autumn

Dashboard with Plotly and Dash

Goal

The goal of this lab session is to get familiar with Dash and Plotly to build simple interactive dashboard visualizations.

Brief Introduction to Dash

Dash is an open-source python framework written on top of Plotly and React for building interactive web-based data visualization applications. It is particularly suited for anyone who want to build web-based data visualization in pure python. Through a couple of simple patterns, Dash abstracts away all of the technologies and protocols that are required to build a full-stack web app with interactive data visualization.

Getting Started

If you have finished the Dash installation and Plotly setting up (Lab 2), you are ready to create your data visualization with Dash and Plotly.

Download the Dashboard folder of Lab 3 from Moodle, then start your Anaconda Prompt (Windows) or Terminal (Mac OS), activate the environment by typing:

```
    conda activate Plotly
```

Navigate to your preferred directory and start the Jupyter Notebook:

```
    jupyter notebook
```

01 - Draw a Treemap Chart with Dash

In this example, we will learn the basic steps for writing dash applications. We will use Treemap to show the daily food sales. Treemap is used to display hierarchical data, it contains a cluster of rectangles with vary size and color, depending on the data value.

A dash application is composed of two parts, **Layout** and **Interaction**.

The **Layout** of an application describes what the application looks like, it mainly consists of two types of components:

- **dash_html_components**: has a component for every HTML tag. For example, the html tag <h1>Hello Dash</h1> can be generated by html.H1('Hello Dash') component.
- **dash_core_components**: describes higher-level interactive components generated with HTML, JavaScript and CSS. In this example, we will use dcc.Graph element to visualize the graphics generated with Plotly.

The **Interaction** is very important in visualization. In Dash you can use callback functions to handle various types of interaction events. We will cover the interactivity of apps in the second example.

Our first step is to import all the packages we need in this example:

```
# necessary libs
# === task 01 ===
import plotly.express as px

# layouts
from dash import html # html tag
from dash import dcc # interactive components
from dash import Dash

import pandas as pd # data
```

Then, we prepare the DataFrame and use the px.treemap() function to link the data to the Treemap chart:

```
dataDict = {
    'food':['chicken', 'pork', 'beef', 'carrots', 'broccoli', 'apples', 'oranges', 'bananas'],
    'sales':[100, 200, 210, 120, 131, 90, 30, 60],
    'category':['Meat','Meat','Meat','Vegetables','Vegetables','Fruits','Fruits','Fruits']
}
df = pd.DataFrame(dataDict)
figTreemap = px.treemap(df, path=['category', 'food'], values='sales')
```

dataDict is a dictionary object, in which each item contains a list of corresponding property. The path parameter in px.treemap() indicates the hierarchy of the Treemap, and the values indicates the property used to specify the area of each region in the Treemap.

Next, create the Dash application and set the corresponding layout:

```
app = Dash()

app.layout = html.Div([
    dcc.Graph(
        id='figTreemap', # one id for one component
        figure=figTreemap
    )
])
```

The html compoent html.Div() is a container, in which we put a dash core compoent dcc.Graph() for rendering the Treemap. Note that, the dcc.Graph() can be used to render any plotly-powered data visualization.

```
app.run() # inline by default
```

Finally, we start the app by typing:

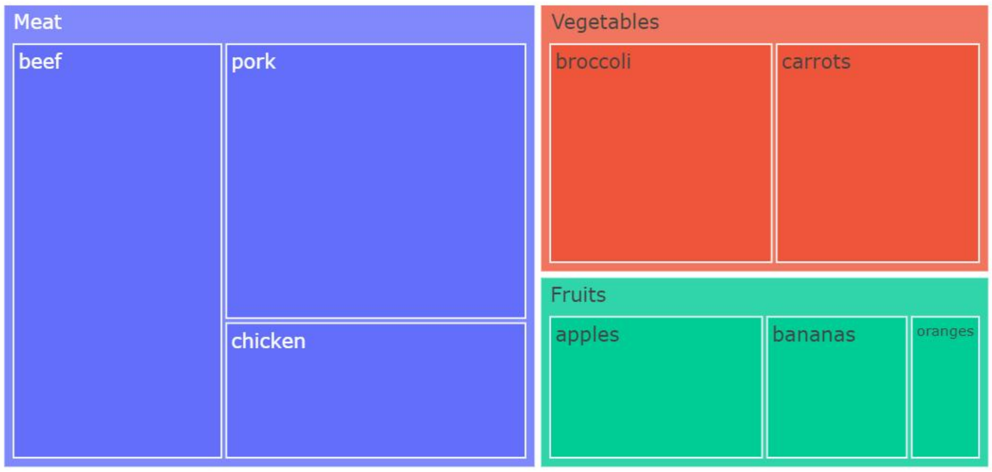
Hit Shift+Enter key to run the script. If everything works well, it will display the visualization directly in the Jupyter notebook output cell by default.

You can also specify other display modes in `app.run(jupyter_mode='external')`. For "external", it will display a URL that you can click to open the app in a browser tab:

```
# display modes: external, tab
app.run(jupyter_mode='external')
```

```
Dash app running on http://127.0.0.1:8050/
```

The visualization will be:



The layout and aspect ratio of the Treemap might be slightly different according to your screen size.

02 - Basic Dash Callbacks for interaction

The basic dash callbacks are a set of python function that are automatically called by Dash when an input event occurs. In this example, we will learn how to handle the **hover event** of the Treemap.

The code in this example is modified from the previous one. To have a clean start, you can new a blank Jupyter notebook and copy all the previous code into the first cell.

First, import additional packages. The Input and Output will be used to describe the input and output of the callback function, and json is used to format the output data.

```
# === task 02 ===
# dash callbacks
from dash.dependencies import Input, Output # input and output of the callbacks
import json
```

Then, add html component html.Pre() with id myoutput to the layout:

```
app = JupyterDash()
app.layout = html.Div(children=[
    dcc.Graph(
        id='figTreemap',
        figure=figTreemap
    ),
    html.Pre(id='myoutput'),
])
```

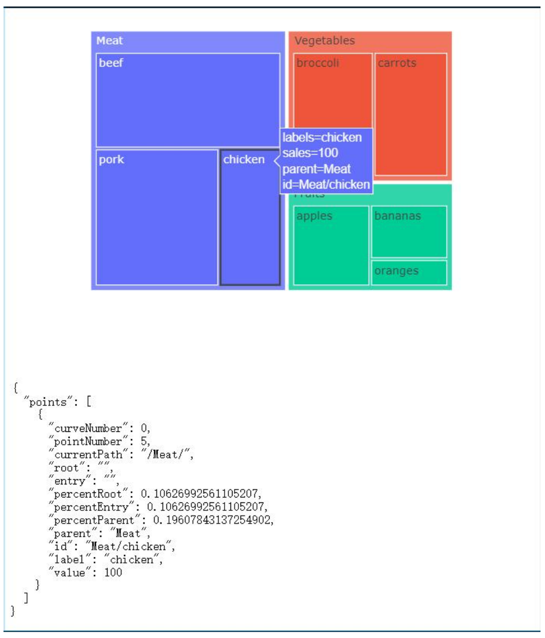
The html.Pre() component can be used to show preformatted text. In this case, it is used to show the content of the hover data.

Next, we add the callback function showHoverData after app.layout = html.Div(…).

```
@app.callback(
     Output('myoutput', 'children'),
    [Input('figTreemap', 'hoverData')]
)
def showHoverData(hoverData):
    return json.dumps(hoverData, indent=2)
```

@app.callback is a decorator, it tells the Dash app to call this function whenever an event is triggered. In our case, the function showHoverData will be called when the component property hoverData of figTreemap is updated, and the returned data of this function will be passed to the children property of the component myoutput. The children property here denotes the content of the html.Pre(). Note that, the @app.callback decorator needs to be directly above the callback function without any blank lines, and you can use any name for the function.

Finally, run the script by hitting Shift+Enter key, and when hovering on the Treemap, the hoverData will be shown below the Treemap:



```
{
  "points": [
    {
      "curveNumber": 0,
      "pointNumber": 5,
      "currentPath": "/Meat/",
      "root": "",
      "entry": "",
      "percentRoot": 0.10626992561105207,
      "percentEntry": 0.10626992561105207,
      "percentParent": 0.19607843137254902,
      "parent": "Meat",
      "id": "Meat/chicken",
      "label": "chicken",
      "value": 100
    }
  ]
}
```

From the hoverData above, we can see that it is a dictionary containing a list of data and the cursor is hovering on the sector named chicken.

03 - Create A Simple Interactive Visualization

In this example, we will learn how to link two charts to create a dashboard. Specifically, a Treemap and a Barchart will be added to the layout, and when hovering on a sector of the Treemap, the corresponding bar will be highlighted.

First, we import the packages.

```python
# === task 03 ===
import copy
```

Compared with the previous examples, this time, we have two more packages: go is used to create the Barchart and copy is a python package for deep copy.

Next, we create a Treemap and a Barchart. And the color of each bar is set to blue:

```python
app = JupyterDash()
dataDict = {
    'food':['chicken', 'pork', 'beef', 'carrots', 'broccoli', 'apples', 'oranges', 'bananas'],
    'sales':[100, 200, 210, 120, 131, 90, 30, 60],
    'category':['Meat','Meat','Meat','Vegetables','Vegetables','Fruits','Fruits','Fruits']
}
df = pd.DataFrame(dataDict)
figTreemap = px.treemap(df, path=['category', 'food'], values='sales')
barColor = ['blue','blue','blue','blue','blue','blue','blue','blue']
figBarchart = px.bar(df, x="food", y="sales", height=400, color_discrete_sequence=barColor)
```

Then, add the Treemap and the Barchart to the layout:

```python
app = JupyterDash()
app.layout = html.Div(children=[
    dcc.Graph(
        id='figTreemap',
        figure=figTreemap
    ),
     dcc.Graph(
        id='figBarchart',
        figure=figBarchart
    )
])
```

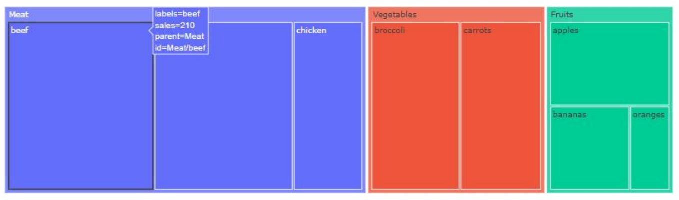
The most important step in this example is to write the callback function LinkTreemapBarchart, which is used to process the hover event of the Treemap and highlight the corresponding bar in the Barchart.

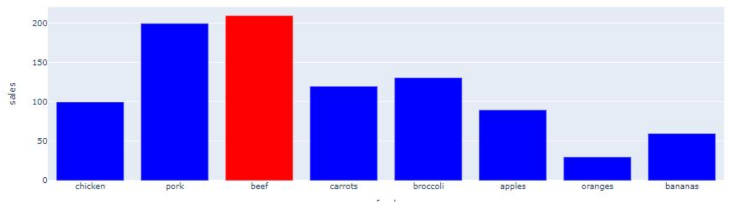
```python
@app.callback(
     Output('figBarchart', 'figure'),
    [Input('figTreemap', 'hoverData')]
)
def LinkTreemapBarchart(hoverData):
    updateBar = copy.deepcopy(figBarchart)
    updateColor = copy.deepcopy(barColor)
    food = dataDict['food']
    if hoverData is not None and 'label' in hoverData['points'][0]:
        # nothing is hovered on when 'label' is not in the dict
        hoverLabel = hoverData['points'][0]['label']
        if hoverLabel in food:
            updateColor[food.index(hoverLabel)]='red'
            updateBar.update_traces(marker_color=updateColor)

    return updateBar
```

Here, the callback function receives hoverData and detects which sector of the Treemap is selected. And then, it updates the color of the corresponding bar in the Barchart.

The visualization will be :



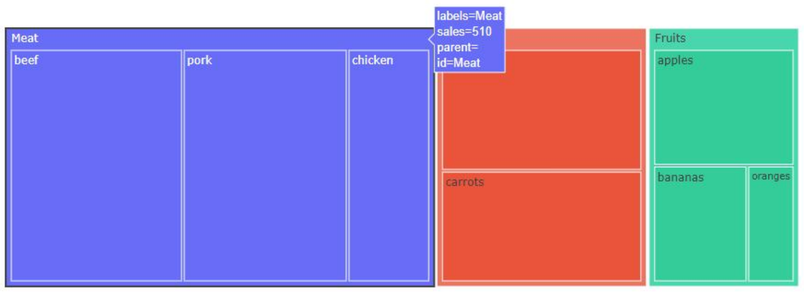


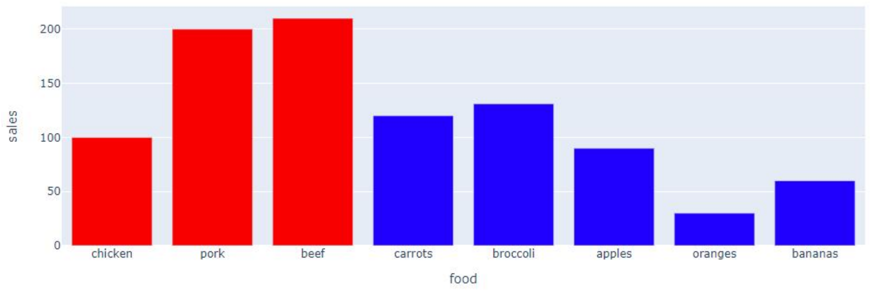
When you hover on a specific part, the same part on both Treemap and Barchart will be highlighted simultaneously.

Tasks for You

Modify the callback function in the third example and achieve the following interaction: When hovering on the category level sector in the Treemap, highlight all the bars in the Barchart corresponding to this category. Please save your visualization result when hovering on the Fruits sector and submit it to Moodle **by Oct 22, 2024**. Remember to add your student number at the end of each figure's title.

The following figure is a sample result when hovering on the Meat sector:


Meat
beef
pork
chicken
labels=Meat
sales=510
parent=
id=Meat
carrots
Fruits
apples
bananas
oranges


sales
food
chicken
pork
beef
carrots
broccoli
apples
oranges
bananas