IN-CORE Data Visualization guide

Forward

This project was completed by graduate student Shahzad Ansari in Summer 2022 under the guidance of Dr.Charles Nicholson at University of Oklahoma. To anyone who plans to use this or maintain it in any of its future iterations, if I am no longer there feel free to contact me. If I am able to help I will.

NOTE: I am by no means an expert at dash (yet) this is just what i have learned so far, i may say things that are technically incorrect just be aware and don't take what i say as gospel.

Installation:

The installation process is pretty straight forward, just follow the commands on this webpage. https://dash.plotly.com/installation

Imports

You will need to install and import the DASH library , use the following imports for almost all DASH applications (in my experience) import more as needed.

```
import pandas as pd
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import dash_table
import math
import numpy as np
```

DASH structure

To create a DASH application you do the following basic steps and customize each to get the result you want.

1) Declare an app by `app = dash.DASH(__name__)`
2) Declaring an app layout by `app.layout = html.Div([],classname="YOURCLASSNAME")` In the Div you set up your html elements
3) Set up your call back, this essentially what controls the responsivity of your dashboard In the way that you can click a graph and change displays etc. you create a callback by `@app.callback()` and in the call back you declare your Output and Input)
4) Under the call back you create a function that uses the input and output of the call back, you need to use every output and input in the function
5) You can have multiple callbacks with each function. You can also chain callback output as the input of the next callback (i didn't need to do this but you can)
6) At the end of the program you need
   ```
   if __name__ == '__main__':
           app.run_server(debug=True,use_reloader=False)
   ```
   Add the flag `use_reloader=False`, if you are using jupyter or some kernel based IDE. It will not work otherwise

Joplin Analysis Archetype

Block 2 Set the color:

In this section I load the `dataframe` and select the desired columns. I then create a new column called `color` and assign a unique color for each `archetype`.

Block 3 Load the app:

`app = dash.Dash(__name__)` before you create a layout you need to instantiate an app. To do with this line of code

Block 4 Create a datatable:

Data Table

https://dash.plotly.com/datatable
Data table is a DASH component used to show information in a interactive table

Parameters

`Id`: 'datatable_id', the id is used to call this component as input when used in callback
`data=dff.to_dict('records')`: puts the data frame into a format that the component can use
`Columns`: the names of each column placed in `[{ " string name " : variable }]`
`editable` : Letting you edit each cell or not, i set to false
`row_selectable`: Set to "multi" , this will create a check box that you can select what rows Of data you want to analyze.
`Row_deletable`: Lets you delete a row, not enabled
`selected_rows` : a array that contains each row that you want selected on creation
`page_action` : If you set the table to a size smaller then all rows you can tab between Tables, this enables that.
`page_current`: first page you want to start on
`page_size` : how many rows in each table
`style_cell_condictionals` : the layout of the table

Drop down bar

https://dash.plotly.com/dash-core-components/dropdown
Creates a drop down bar where you can select a value from

Parameters

`Id`: Unique id for this dropdown.
`options`: When you click on the drop down the options within it are defined by a list containing values in the format {'label' : labelstring, 'value':'df column name'}
`value`: Starting default value.
`multi`: be able to select multiple values from dropdown, default as false.
`clearable`: be able reset the value selected to null, default as false.

Graph

https://dash.plotly.com/dash-core-components/graph
The component used to show graphs and visualizations, think of this as declaring that this html container will contain a graph of some sort. You haven't said what kind just yet but its going to be a graph.

Parameters

`Id`: The name of the graph , used for output in callback

Block 5 The callback:

Callback

The first part is the callback which is started by `@app.callback( )`. In the parenthesis you put in the outputs and inputs.
https://dash.plotly.com/basic-callbacks

Output: `[Output('graphs', 'figure'), Output('map','figure')]` : here i am saying that there are two outputs named 'graph' and 'map' (see Block 5 graph) and they are of type 'figure'.

Input: `[Input('chartdropdown', 'value'), Input('datatable_id','selected_rows'), Input('datatable_id', 'data')]` : Here I am saying that there are 3 inputs. `chartdropdown` is going to give us a `value` which is a parameter of `dropdowns`. `datatable_id` will give us a `selected_rows` and `data` which are parameters of datatable.

NOTE: you technically don't need to surround in/output with [ ] but sometimes it wont work unless you do. I'm not sure why this is the case.

Callback function

Every callback needs to be followed with a function, technically you can have more then one function (I believe) But they need to also use all the inputs and outputs within the callback.

Parameters

It takes in `chartdropdown, selected_rows, data,` the input from the callback.

P1. Selected the specific archetypes

In this section i set up the mechanism to get which archetypes the user selects from the data table, and retrieve the rows from the data frame corresponding to the archetypes. The structure is essentially in pseudo code:

```
If there are no selected rows
     Select every row
     Get the data frame containing the information from every row
Else
     For every row that is selected
     Get the data frame containing the information on the selected
row
```

P2. Make the facet figure

https://plotly.com/python/subplots/

This is how you make a graph with subplots within it, essentially it is a grid with user specified dimension and in each of the row,column intersections you put in a plot

Parameters

`rows`: specify the amount of rows

`cols`: specify the amount of columns

`Column_width`: specify the size of each of the columns like `[ size1,size2,size3]` if `columns = 3`

`row_width`: specify the size of each of the columns like `[ size1,size2,size3]` if `rows = 3`

P3. Histogram

https://plotly.com/python/histograms/

Parameters

`x:` the data you pass for the histogram

`texttemplate:` the format you want to display the text or what string you want

`textfont_size:` the size of the text

P4. Box Plot

https://plotly.com/python/box-plots/

You will see a line `fig.add_trace(Histogram_1,1,1)` what this is saying is that, to that figure i made in P2. on row 1 column 1 i will add a histogram.

`fig.update_xaxes(range=[lowerRange_hist-1,upperRange_hist+1],type="linear")` i set the range and scale of the x axis.

```
fig.update_layout(title_text=f"Information on {chartdropdown}",showlegend=False,width=1000,height=1000,bargap=0.2)
```

Add the text and set the dimensions of the figure.
Parameters
`x`: the data similar to P3.

P6. Map Box

Firstly you need to make a mapbox access token, you can do so here
https://docs.mapbox.com/help/getting-started/access-tokens/

The primary function that you will use for the map is `go.Figure(go.Scattermapbox( ))` found here
https://plotly.com/python/scattermapbox/

Parameters

`lat`: pass in the list of latitude coords
`lon`: pass in the list of longitude coords
`mode`: the type of symbol on the map, this case is a marker
`marker`: specify the color and size of the marker

P7. Map Layout

This section I specify the layout of the map, more information is here.
https://plotly.com/python/map-configuration/

Parameters

`autosize:` automatically size the map
`hovermode:` what happens when you hover near a marker
`showlegend:` legend of the colors on the map
`mapbox:` A dictionary containing the

- `accesstoken:` The token you generated to access mapbox
- `bearing:` The direction the map starts facing.
- `center` : a dictionary containing the
    - `lat`: the starting latitude
    - `lon`: the starting longitude

- `pitch`: the angle you look at the map
- `zoom`: how close in you look at the map

`width:` The width of the mapbox
`height:` The height of the mapbox

P8. Return the output

I return a list of both the figure and map, this corresponds to the output from Callback.

Block 6: Run the server

This is the section that actually runs the server itself

preato_analysis

Block 1:

See Imports

Block 2: Loading Dataframe

In this portion load the data frame, select the desired columns and rows. I fix the naming convention and rename the `Blockgroup` column then cast it to `int`.

Block 3: Solution XLS

To load a xls file with multiple sheets use the function `pd.ExcelFile()` and to get each sheet `pd.read_excel(dataframe,'sheetname')`. I load the xls then cast the `Blockgroup` into `int`.

Block 4: Data Manipulation

In this section i need to get the centroids of the coordinates, the count of each type of block group and the mode. I do this by finding each in an individual dataframe then merging them.

Probably not the most efficient method as I'm technically creating 3 dataframe's that don't need to exist but it works.

Block 5: Mitigation Color

I then set a unique color for each mitigation_level similar to how I set the color in Joplin Analysis. After that I then delete the unused dataframe's to free up memory .

Block 6: Load the app

See loading app.

Block 7: 3D Scatterplot

First I create a figure object, and customize the type of figure by `fig.add_trace()` to that I pass in the go.Scatter3d(). To read more https://plotly.com/python/3d-scatter-plots/

go.Scatter3d()

Parameters

`x:` The list of x values for the 3d scatterplot
`y:` The list of y values for the 3d scatterplot
`z:` The list of z values for the 3d scatterplot

`mode:` The type of markers on the scatter plot, the symbols.
`name:` The name of the markers so you can customize them
`markers:` From the name, set it to a dictionary with these parameters

- `size:` Size of the markers
- `color:` Color of the markers, can be static color or dynamic when passing in a list of values and a color scale.
- `colorscale:` This is a premade color scale as the values in color increase this dictates the change in color
- `opacity:` The opacity of each marker.

Block 8: Figure interactivity

In this section I set up that when I click a marker an event occurs and I select the marker, this returns a json object. After this I set the starting axis labels and dimensions of the figure. See https://plotly.com/python/reference/layout/ for more information.

Block 9: Layout

Within the `app.layout = htmlDiv([ ])`, i create a graph passing in the figure object created in Block 7 similar to graph.

Block 10: Callback

See Joplin Callback. This case i do not place input and output in brackets.
Note that the returned obj from the figure goes from the figure to the graph and in the callback it is used as input called clickData

Block 11: Callback Function

This function exists to updated the map as you select points on the preato frontier. You need to create your own mapbox token and set `mapbox_access_token` to it.

P1. Render Map when no point is clicked

See Joplin Map Box.

P2.Render map when click data

We need to get the x, y and z coordinates from the the pareto frontier, the way to index information form clickData `clickData['parameter'][i]['parameter']` in this case as an example `xCoord = clickData['points'][0]['x']` we get the points and the 0th index and the x value.

After that I find what `solution_ID` correlates to the `x, y` and `z` values. Extract a dataframe from the `modeDF` that has only rows of that `solution_ID` and call it sample . After that I map the sample in the same way here. Note that the `marker` parameter in this case i made the size scalable with the count of each `Blockgroup`.

P3. Map Layout

See Joplin Map Layout

Block 11: Run the server

See Joplin running app.

Additional Resources

https://www.youtube.com/c/CharmingData Amazing youtuber very useful for dash
https://dash.plotly.com/ The documentation for DASH
https://www.tutorialspoint.com/python_web_development_libraries/python_web_development_libraries_dash_framework.htm Some helpful documents

Contact Information

Author: Shahzad Ansari
Email: shahzad1611@gmail.com
Advising Professor: Dr. Charels Nicholson
Date: 8/5/2022
Institution: University of Oklahoma