Using Dash for Visualisations

https://realpython.com/python-dash/

Notes from plotly tutorial: https://dash.plotly.com/tutorial

Examples with plotly : https://dash-example-index.herokuapp.com

```
# Import packages
from dash import Dash, html

# Initialize the app
app = Dash(__name__)
```

The app layout represents the app components that will be displayed in the web browser, normally contained within a `html.Div`. In this example, a single component was added: another `html.Div`. The Div has a few properties, such as `children`, which we use to add text content to the page: "Hello World"

```
# App layout
app.layout = html.Div([
    html.Div(children='Hello World')
])
```

These lines are for running your app, and they are almost always the same for any Dash app you create.

```
# Run the app
if __name__ == '__main__':
        app.run(debug=True)
```

Including data

```
# Import packages
from dash import Dash, html, dash_table
import pandas as pd

# Incorporate data
df =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv'
)

# Initialize the app
app = Dash(__name__)

# App layout

app.layout = html.Div([
                                        html.Div(children='My First App with Data'),
                                        dash_table.DataTable(data=df.to_dict('records'),
page_size=10) ])

# Run the app
```

```
if __name__ == '__main__':
        app.run(debug=True)
```

Plotting

Use Plotly seems to be the way.
Plotly chart types

- Include like any other element in the app layout

Controls and Callbacks

So far you have built a static app that displays tabular data and a graph. However, as you develop more sophisticated Dash apps, you will likely want to give the app user more freedom to interact with the app and explore the data in greater depth. To achieve that, you will need to add controls to the app by using the callback function.

In this example we will add radio buttons to the app layout. Then, we will build the callback to create the interaction between the radio buttons and the histogram chart.

```
Controls and Callbacks

So far you have built a static app that displays tabular data and a graph. However, as
you develop more sophisticated Dash apps, you will likely want to give the app user more
freedom to interact with the app and explore the data in greater depth. To achieve that,
you will need to add controls to the app by using the callback function.

In this example we will add radio buttons to the app layout. Then, we will build the
callback to create the interaction between the radio buttons and the histogram chart.
# Import packages
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px

# Incorporate data
df =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv'
)

# Initialize the app
app = Dash(__name__)

# App layout
app.layout = html.Div([
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'], value='lifeExp',
id='controls-and-radio-item'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
])

# Add controls to build the interaction
@callback(
    Output(component_id='controls-and-graph', component_property='figure'),
    Input(component_id='controls-and-radio-item', component_property='value')
```

```
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

Layout

- Can be specified with

HTML and CSS

HTML and CSS are the lowest level of interface for rendering content on the web. The HTML is a set of components, and CSS is a set of styles applied to those components. CSS styles can be applied within components via the `style` property, or they can be defined as a separate CSS file in reference with the `className` property, as in the example below.

Dash Bootstrap Components

Dash Bootstrap is a community-maintained library built off of the bootstrap component system. Although it is not officially maintained or supported by Plotly, Dash Bootstrap is a powerful way of building elegant app layouts. Notice that we first define a row and then the width of columns inside the row, using the `dbc.Row` and `dbc.Col` components.

For the app below to run successfully, make sure to install the Dash Bootstrap Components library: `pip install dash-bootstrap-components`

Theme explorer **https://dash-bootstrap-components.opensource.faculty.ai/docs/themes/explorer/**
This one is also nice https://hellodash.pythonanywhere.com

Plotly Dash Design Kit (DDK) (NEEDS LICENCE)

Dash Design Kit is our high level UI framework that is purpose-built for Dash. With Dash Design Kit, you don't need to use HTML or CSS. Apps are mobile responsive by default and everything is themeable. Dash Design Kit is licensed as part of Dash Enterprise and officially supported by Plotly.

Here's an example of what you can do with Dash Design Kit (note that you won't be able to run this example without a Dash Enterprise license).

Managing large data

If you don’t have much computing resources, you can choose to size down the json or shapefile by following the instructions in this link.

Interactive maps

```
pip install dash==2.12.1
pip install dash-leaflet==1.0.0
```

dash-leaflet is better than folium

Better integration with Dash

Possible tools

- **Geo maps:** Plotly's outline-based mapping from Plotly Express or Plotly Graph Objects
- **Mapbox maps**: Tile-based basemaps from Mapbox
- **Dash Leaflet**: A Dash component built around the popular lightweight Javascript library
- **Dash Deck**: A proof-of-concept Dash component built on top of deck.gl, known for supporting 3D visualization and large datasets.
- **Datashader:** While not necessarily just for geospatial data, Datashader can help you display your big data on a map.