Plotly's Dash

MODULE 10

What is plotly?

- A Python library for creating modern, interactive graphs
  - Wraps JavaScript but code in Python
- **plotly.express** for graphs
- for more refer to : https://colab.research.google.com/drive/1wUaWf-CK-OmhWJQSrFRWguHsrEJPM_Wk?usp=sharing
- **customizing Plotly graphs:**
  - plotly graph properties can be updated later with update_layout()
  - eg: Choosing the bar width of our bar graph
    - *bar_fig.update_layout({'bargap': 0.5})*
    - *bar_fig.show()*
- Check out plotly documentation at: https://plotly.com/python-api-reference/

From Plotly to Dash

- Introduction to Dash
  - Dash is the low-code framework for rapidly building data apps.
  - It is written on top of Plotly.js and React.js.
  - It is ideal for building Data apps with cusomized user interfaces.
  - suitable to the people who work with data.
  - Dash apps are rendered in the web browser.

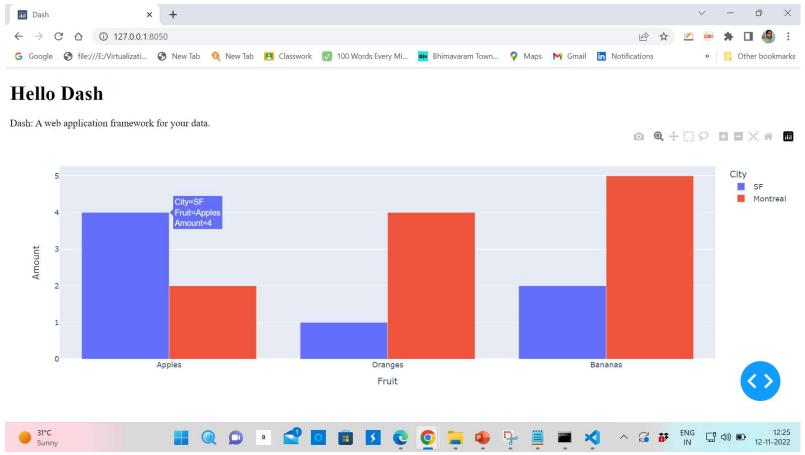
From Plotly to Dash

- **Dash Installation**
  - to install dash
    - **pip install dash**
  - This installation also brings along the plotly graphing library. This library is under active development, so install and upgrade frequently.
  - If you prefer Jupyter notebook, install jupyter-dash
    - **pip install jupyter-dash**

- Dash apps are composed of two parts.
  - The first part is the **"layout"** of the app and it describes what the application looks like.
  - The second part describes the **interactivity** of the application

- ***Ready? let's make your first Dash app!***

A first Dash app




```python
from dash import Dash, html, dcc
import plotly.express as px
import pandas as pd

app = Dash()

df = pd.DataFrame({
    "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"],
    "Amount": [4, 1, 2, 2, 4, 5],
    "City": ["SF", "SF", "SF", "Montreal", "Montreal", "Montreal"]
})

fig = px.bar(df, x="Fruit", y="Amount", color="City", barmode="group")

app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for your data.
    '''),

    dcc.Graph(
        id='example-graph',
        figure=fig
    )
])
```

```
$ python app.py
...Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

Dissection of the first app

- Dash is the main library that creates the app itself.
- The layout is composed of a tree of components
  - eg: html.Div and dcc.Graph
- **dash.html** is the Dash HTML Components module
  - It has a component for every HTML tag.
  - The *html.H1(children='Hello Dash')* component generates a *<h1>Hello Dash</h1>* HTML element in your application
- **dash.dcc** is the Dash Core Components module.
  - It contains higher-level components that are interactive.
  - Each component is described entirely through keyword attributes.
- The children property is special. By convention, it's always the first attribute which means that you can omit it: *html.H1(children='Hello Dash')* is the same as *html.H1('Hello Dash')*.

The app layout

```
app = dash.Dash()
app.layout = dcc.Graph(
  id='example-graph',
  figure=bar_fig)
```

- Create an app object using `dash.Dash()`
- Set the `app.layout`
  - Here, a single graph
  - Using `dcc.Graph()`
    - `figure` = The Plotly figure to render
    - `id` = Important for callbacks later

Running the app

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

- Lastly, running the server
- Script is run from command-line (not read into a notebook)
  - i.e., `python my_app.py` in the command-line
- `debug` for helpful feedback when testing

Script is run via the command-line (`python3 script.py`), served on a local server

Access via a web browser such as Google Chrome

While served, update and save `.py` file to see live updates in browser

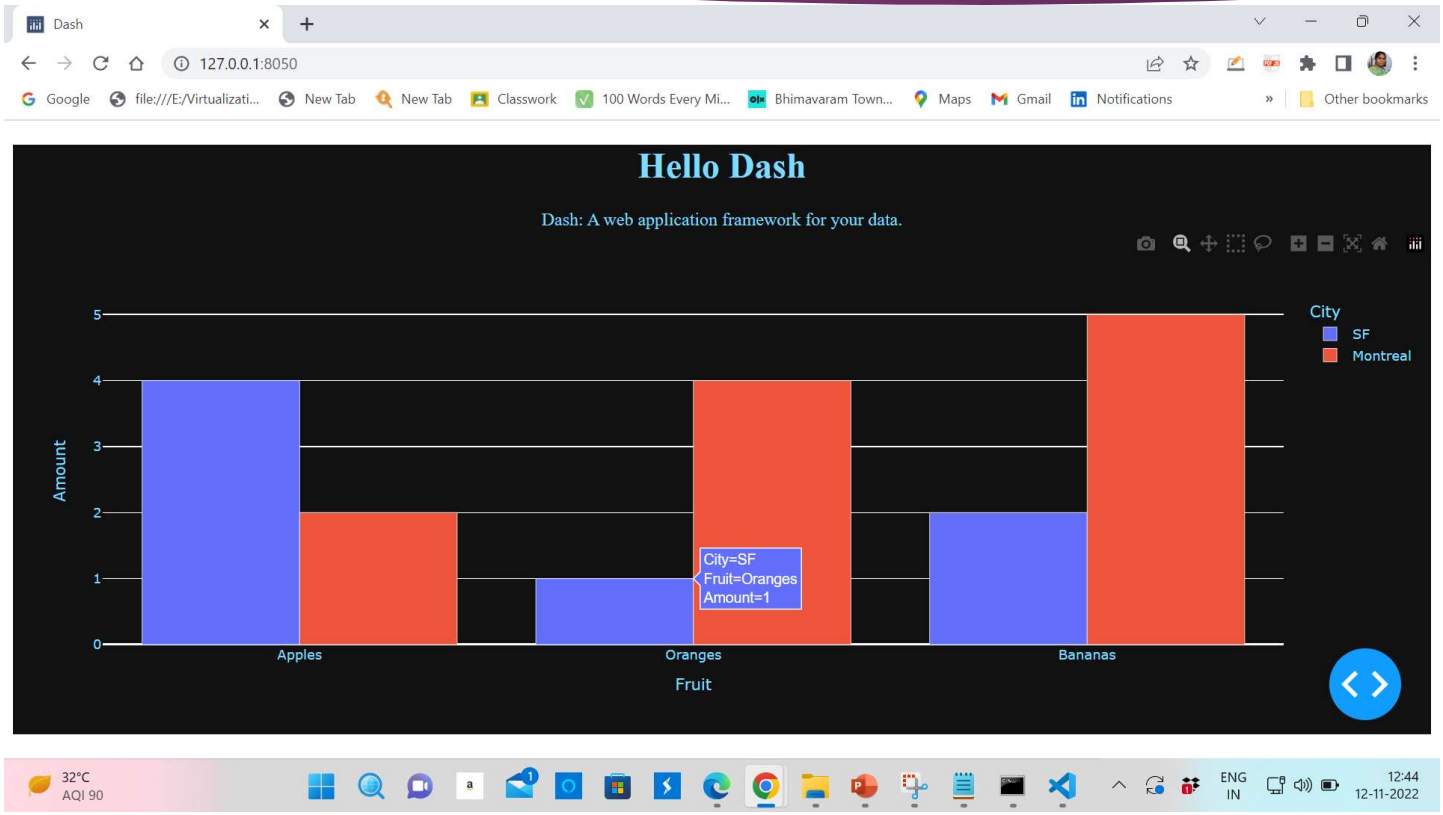
```
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app "simple_app" (lazy loading)
 * Environment: production
```

Hot-reloading

- Dash includes "hot-reloading", this features is activated by default when you run your app withapp.run_server(debug=True).
- This means that Dash will automatically refresh your browser when you make a change in your code.
- Try it out

Update the first app by adding
- background color and text color.
- add inline styles.
- center the headings.

It should look like the one below.



Shall we do this now?

More about HTML components

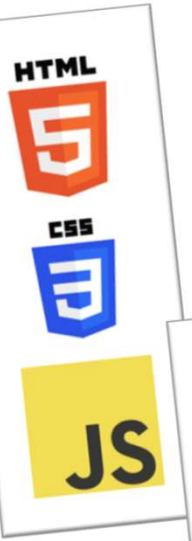
- Dash HTML Components contains a component class for every HTML tag as well as keyword arguments for all of the HTML arguments.
- In our example, we modified the inline styles of the html.Div and html.H1components with the style property.
  - html.H1('Hello Dash', style={'textAlign': 'center', 'color': '#7FDBFF'})
  - This code is rendered in the Dash application as <h1 style="text-align: center; color: #7FDBFF">Hello Dash</h1>.
- There are a few important differences between the dash.html and the HTML attributes:
  - The style property in HTML is a semicolon-separated string. In Dash, you can just supply a dictionary.
  - The keys in the style dictionary are camelCased. So, instead of text-align, it's textAlign.
  - The HTML **class** attribute is **className** in Dash.
  - The children of the HTML tag is specified through the children keyword argument. By convention, this is always the first argument and so it is often omitted.
- Besides that, all of the available HTML attributes and tags are available to you within your Python context.

Positioning Dash Components

HTML and the web

HTML: language for structuring websites

- HTML: wooden structure of a house
  - Set placement of objects
- CSS: paint color of a room
  - Style (e.g., background color) of objects
- JavaScript: Smart home clap-on lights!
  - Interactivity e.g., clickable actions



Div and H tags

Dash uses `dash_html_components` to interface between HTML and Python

Two important HTML structures ('tags'):

- Div tags: important for structuring websites
  - Can have many different-sized divs with different things inside
- H tags: different sized titles (H1 > H6)

Using Div and H tags

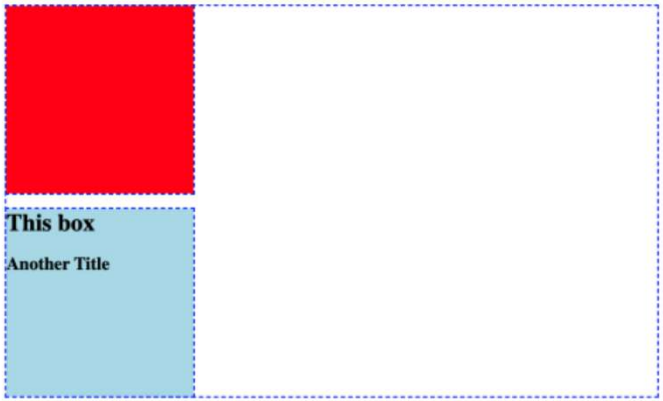
Some HTML code with:

- Overall div (everything inside)
- Div inside: red background
- Div with blue background
  - H tags inside
- Ignore the `style` part - more on 'CSS' later!

```
<div>
  <div style="background-color: red;
              width:250; height:250;">
  </div>
  <div style="background-color: lightblue;
              width:250; height:250;">
    <h1>This box</h1>
    <h2>Another Title</h2>
  </div>
</div>
```

See this example in Dash in dashboard2.py

Our example



Take note:

- Red background div
- Blue background div with H tags

The div tag can nest; lots of possibilities when structuring our web app.

Displaying a dataframe as a table

- By writing our markup in Python, we can create complex reusable components like tables without switching contexts or languages
- A quick example is present in table.py

What is CSS?

CSS stands for 'Cascading Style Sheets'

- A language to determine how a webpage is styled
- We can control:
  - Text/font properties
  - Size & shape of objects (HTML tags!)
  - Placement of objects

CSS on the web

Most websites have CSS files;

- They are read in on page load

CSS on HTML elements

CSS can also be used on a HTML element

- Use the `style` property of a tag
- CSS statements in one big string separated by `;`
  - Statements are `property:value;property:value;`

```
<div>
  <div style="background-color: red;
              width:250; height:250;">
  </div>
  <div style="background-color: lightblue;
              width:250; height:250;">
    <h1>This box</h1>
    <h2>Another Title</h2>
  </div>
</div>
```

CSS in Dash

- Dash components `style` argument
  - Accepts CSS as a dictionary
- Previous code in a Dash component:

```
app.layout = html.Div([
  html.H1('Welcome to the website!'),
  html.H2('Text',
  style={'font-size':'50px',
         'color':'red'}
        )])
```

A border on our app

```
html.Div(dcc.Graph(figure=ecom_bar),
 style={'width':'500px',
        'height':'450px',
        'border':'5px dotted red'}
```

CSS for layout

Elements not aligning? HTML elements can either be 'inline' or 'block' elements.

- Inline render on the same line
  - Have no height or width (or box) properties
  - Examples include , <a>, <img>
- Block stack on top of one another as they include a line break
  - Can't have more than one side-by-side
  - Examples include <h1>,
- There is also inline-block
  - Can set height/width/box properties
  - Can be side-by-side

Dash core components (Revisiting)

- Dash Core Components (dash.dcc) includes a set of higher-level components like dropdowns, graphs, markdown blocks, and more.
- Like all Dash components, they are described entirely declaratively.
- Every option that is configurable is available as a keyword argument of the component.
- Refer to documentation for all the core components
- A few of the components can be examined in a file named corecomponets.py (Examine the code)

Callbacks

- **What are callbacks?**
  - Callbacks are the functionality triggered by interaction.
  - A user interacts with an elements --> A Python function is triggered --> Something is changed.
- **Why callbacks?**
  - To enhance the interactivity.
- **Callbacks in Dash**
  - Start with a decorator function
  - uses ***from dash import Input, Output***
  - **Output**: Where to send the function return
    - component_id: Identify the component
    - component_property: What will be changed
  - **Input**: What triggers the callback
    - component_property: What to use in triggered function.

```
@app.callback(
    Output(component_id='my_plot',
           component_property='figure'),
    Input(component_id='my_input',
           component_property='value')
)
def some_function(data):
    # Subset Data
    # Recreate Figure
    return fig
```

Understand the following code....

```
app.layout = html.Div(children=[
  dcc.Dropdown(id='title_dd',
      options=[{'label':'Title 1',
                'value':'Title 1'},
               {'label':'Title 2',
                'value':'Title 2'}]),
  dcc.Graph(id='my_graph')])
@app.callback(
    Output(component_id='my_graph',
           component_property='figure'),
    Input(component_id='title_dd',
           component_property='value')
)
```

```
#@app.callback()
def update_plot(selection):
    title = "None Selected"
    if selection:
        title = selection
    bar_fig = px.bar(
      data_frame=ecom_sales,
      title=f'{title}',
      x='Total Sales ($)', y='Country')
  return bar_fig
```

Dropdown as a filter

- Common use case - dropdown filters the plot DataFrame

```
#@app.callback()
def update_plot(input_country):
    input_country = 'All Countries'
    sales = ecom_sales.copy(deep=True)
    if input_country:
        sales = sales[sales['Country'] == input_country]
    bar_fig = px.bar(
        data_frame=sales, title=f"Sales in {input_country}",
        x='Total Sales ($)', y='Country')
    return bar_fig
```

Enhancing the interactivity

- Some useful interactive components:
  - `dcc.Checklist()` = Checkboxes
  - `dcc.RadioItems()` = Radio buttons
  - `dcc.Slider()` / `dcc.RangeSlider()` = Slider selectors
  - `dcc.DatePickerSingle()` / `dcc.DatePickerRange()` = Similar to sliders but for dates

Sliders

- Slider: drag and move for a single value

A slider:

A range slider:

- Range Slider: drag and move for two values
- Reminder: Can link to callback
  - Update plots or components

Sliders in Dash

```
dcc.Slider(
    min=10,
    max=50,
    value=45,
    step=5,
    vertical=False
)
```

Key arguments:

- `min` / `max` : Bounds of slider
- `value` : Starting selection
- `step` : Increment for each notch
- `vertical` : To make horizontal or vertical

Date pickers in Dash

`DatePickerSingle` : Select a single date

```
dcc.DatePickerSingle(
  date=date(2021, 7, 1),
  initial_visible_month=datetime.now(),
)
```

- `date` = starting selection
- `initial_visible_month` = month shown in popup
- Optionally limit `min_date_allowed` and `max_date_allowed`

Date Range Picker

- Similar to `DatePickerSingle`

```
dcc.DatePickerRange(
initial_visible_month=datetime.now(),
start_date=date(2021, 7, 1),
end_date=date(2021, 7, 14),
)
```

- Set an initial `start_date` and `end_date`